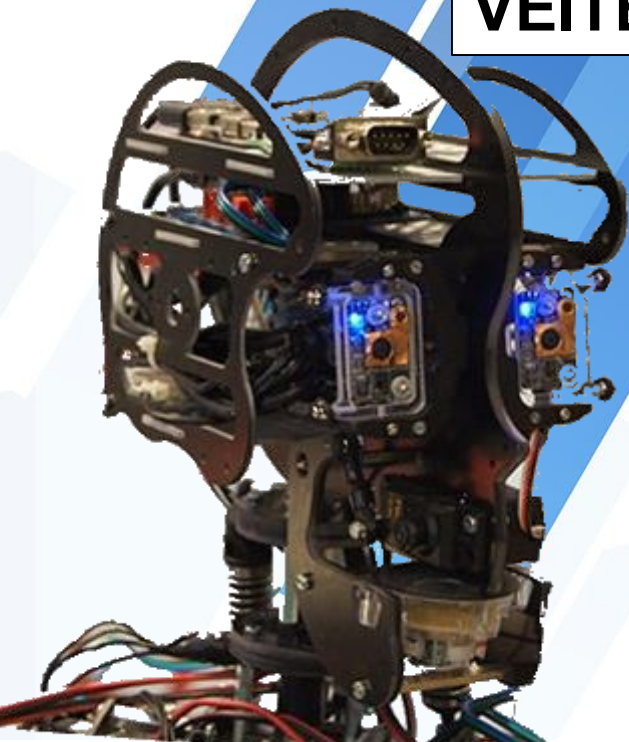


**BTS IRIS
Lycée Turgot
Limoges**

Projet Module de tête autonome

Humanoïde « Aria »

**CHAUMETTE Florent
LOUIS Nicolas
STØEBNER Kévin
VEITES Thomas**



2014 - 2015

Table des matières

I.	Partie commune	4
1.	Rappel du Cahier des charges	4
a.	Présentation du projet	4
b.	Expression du besoin	4
2.	Capture des besoins	6
a.	Contexte du système	6
b.	Ensemble des cas d'utilisations	6
c.	Diagramme de séquence d'initialisation de la tête	7
d.	Diagramme de déploiement	8
3.	Analyse et conception préliminaire	9
a.	Diagramme de classe	9
b.	Diagramme Etat-transition	10
II.	VEITES Thomas : Reconnaissance vocale et réalisation de l'I.H.M.	11
1.	Rôle dans le projet	11
a.	Reconnaissance Vocale	11
b.	Intégration des différentes Classes	11
c.	Interface Homme Machine	11
2.	Mise en place	11
a.	BeagleBone Black	11
b.	Installation d'un SE sur BeagleBone	11
c.	Configuration réseau de la BeagleBone	12
d.	Mise en place des utilisateurs	12
e.	Installation d'un serveur SSH	12
f.	Installation interface graphique	13
3.	Reconnaissance Vocale	13
a.	A propos de PocketSphinx	13
b.	Installation PocketSphinx	13
c.	Utilisation en C++	14
d.	Classe Ears	14
e.	Classe Brain	15
f.	Relations entre les classes	16
g.	Intelligence Artificiel	17

4.	Le Framework Qt4	17
a.	A propos de Qt4	17
b.	Installation de Qt4	17
c.	Présentation de L'Interface Homme Machine	18
d.	Compilation de l'IHM sur BeagleBone Black	19
e.	Problèmes rencontrés	20
5.	Conclusion	20
III. CHAUMETTE Florent : Suivi et reconnaissance faciale.....		21
1.	Situation dans le projet	21
2.	OpenCv	21
a.	Introduction.....	21
b.	Installation sur PC GNU/Linux Debian 7	22
c.	Prise en main d'OpenCv	22
3.	Cas d'utilisation « Détecter les visages »	26
a.	Capture du besoin	26
b.	Analyse et Conception préliminaire	26
c.	Mise en place de l'environnement du système	28
d.	Conclusion sur l'avancement du travail	30
4.	Cas d'utilisation : « Suivre une personne »	31
a.	Capture des besoins	31
b.	Analyse et Conception préliminaire	31
c.	Mise en place de l'environnement du système	31
d.	Conclusion sur l'avancement du travail	32
5.	Cas d'utilisation : « Reconnaître une personne ».....	32
a.	Capture des besoins	32
b.	Analyse et Conception préliminaire	32
c.	Mise en place de l'environnement du système	32
d.	Conclusion sur l'avancement du travail	34
6.	Classe Eye / Eyexception	35
a.	Diagramme (UML)	35
b.	Explications.....	35
c.	Version Lite.....	36

IV. LOUIS Nicolas : Commande des servomoteurs RC et Conversion « Text to Speech »	37
1. Rôle dans le projet.....	37
2. Planification du projet	37
3. La compilation croisée.....	38
a. Qu'est-ce que la compilation croisée ?	38
b. Comment la mettre en place ?	38
c. Comment l'utiliser ?	38
4. Carte MicroMaestro du constructeur Pololu	39
a. Qu'est-ce que la Pololu ?.....	39
b. Quels sont les modes de connexion ?	39
c. Les Servomoteurs de type RC.....	40
d. La classe ServoMoteurRC	43
a. Qu'est-ce que le Text-To-Speech ?.....	43
e. La classe Machoire	43
f. Tests unitaires	44
5. ESpeak	46
a. Contrainte de la BeagleBone Black	46
b. La classe CVoix.....	47
c. Tests unitaires	47
6. Conclusion	48
V. STOEBNER Kévin : Commande des servomoteurs Dynamixel.....	49
1. Rôle dans le projet.....	49
2. Servomoteur Dynamixel : MX28-T	49
a. Présentation du servomoteur	49
b. Communication avec les servomoteurs	50
3. Configuration.....	51
a. Configuration des servomoteurs.....	51
b. Configuration de l'USB2AX	52
4. Conception des classes ServoDyn et Neck	53
a. Classes ServoDyn et classe mère ServoMoteur	53
b. Classe Neck	54
c. Conclusion	55
Webographie	56

I. Partie commune

1. Rappel du Cahier des charges

a. Présentation du projet

Le but du projet est de rendre un module de tête de la société Cybedroid autonome. Actuellement ce module de tête est piloté par une application fonctionnant sous Microsoft Windows. Cette application pilote les différents servomoteurs de la tête (trois pour le cou, quatre pour l'orientation des caméras, un pour la mâchoire), génère la synthèse vocale, affiche le flux vidéo des caméras.

Il s'agit d'assurer toutes ces fonctions à l'aide d'un système embarqué fonctionnant sous GNU/Linux. Des fonctions supplémentaires seront aussi implantées sur cette plateforme :

- reconnaissance vocale,
- interaction avec l'utilisateur,
- reconnaissance et suivi d'une personne ou d'un objet simple,
- diffusion d'un flux vidéo 3D construit à partir des deux caméras...

Un écran tactile couleur permettra d'interagir avec un utilisateur afin d'analyser et tester tous les périphériques présents sur le module de tête.

b. Expression du besoin

Le module de tête existe et est actuellement commercialisé. Un bus série TTL half-duplex permet de piloter les trois servomoteurs Dynamixel intelligents du cou. L'interface entre le bus USB du PC de pilotage de la tête et le bus série TTL est réalisé avec un module USB2AX¹. Les servomoteurs des yeux et de la bouche sont des modèles standards (commandés par une modulation de largeur d'impulsion). Ils sont pilotés depuis un port USB du PC de pilotage par un module Pololu MicroMaestro². Les yeux sont des webcams HD USB de marque Hercules. Le son est amplifié en local par un amplificateur puis restitué par le haut-parleur qui se trouve dans la bouche. L'énergie est fournie par un bloc alimentation.

Une seconde version de la tête utilise des servomoteurs Dynamixel sur bus RS485, pour dialoguer avec ces servomoteurs nous utiliserons un convertisseur USB2Dynamixel³.

Ces servomoteurs sont qualifiés d'intelligent car ils embarquent un microcontrôleur et une mémoire EEPROM qui permettent de :

- réaliser des commandes complexes du moteur : régulateur PID réglable pour piloter la position et/ou la vitesse,
- de sauvegarder les paramètres du servomoteur comme par exemple l'identifiant sur le bus,
- d'assurer un dialogue avec un système de commande (Ex : PC) par l'intermédiaire d'un bus série. Le dialogue utilise une trame propriétaire décrite par le constructeur⁴.

On souhaite que le robot reconnaisse et exécute des ordres vocaux simples. Pour cela il devra capter le son de la voix par deux micros USB (simulant les oreilles) puis effectuer une reconnaissance vocale (conversion de la parole en texte). Ce texte sera analysé pour générer une réponse adapté tant au niveau vocale (conversion texte vers parole) que gestuel (mouvement de la tête).

¹<http://www.xevel.fr/blog/index.php?post/2011/08/31/USB2AX>

²<http://wiki.cybedroid.com/index.php?title=MicroMaestro>
http://wiki.cybedroid.com/index.php?title=Protocole_maestro_Pololu

³http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm

⁴http://wiki.cybedroid.com/index.php?title=Protocole_dynamixel_Robotis

La tête devra être capable d'identifier une personne présente devant elle préalablement enregistrée puis de suivre visuellement cette personne lors de ces déplacements. Un objet coloré pourra remplacer une personne (balle rouge par exemple). La localisation d'une personne pourra aussi se faire par la voix.

Un écran tactile intégré dans la tête doit permettre d'afficher des informations relatives au système (par exemple pendant la phase de développement) et faciliter l'analyse du matériel présent et le test de celui-ci.

Compte-tenu de ce qui précède les plateformes GNU/Linux embarqué retenues sont :

- une carte RaspberryPi⁵ modèle B ou B+ avec une distribution Raspian,
- une carte BeagleBoneBlack⁶ de Texas Instrument avec une distribution Debian.

Le module de tête doit interpréter des ordres simples et répondre de manière adapté comme par exemple :

- ordre vocal : bonjour → réponse vocal : bonjour associé à un mouvement de la bouche et de la tête,
- ordre vocal : date → réponse vocal : date du jour, mouvement de la tête
- ordre vocal : heure → réponse vocal : heure courante, ...
- etc...

Le système de reconnaissance vocal pourra être issu du projet CMU Sphinx⁷. Un fichier texte contenant les mots clés reconnus sera généré automatiquement. Un logiciel « analyseur » lira ce fichier et en fonction des mots clés qu'il y aura trouvés, un autre fichier texte sera généré avec la réponse.

La génération vocal pourra être confiée au projet espeak⁸ ou mbrola⁹ voire l'association des deux. La réponse vocale sera issue du fichier texte généré par l'analyseur.

La reconnaissance de forme (par exemple une balle), de visage sera confiée à la bibliothèque OpenCV¹⁰. Ainsi une fois que l'on a reconnu une personne ou une forme, la tête pourra suivre l'objet en mouvement dans les limites physiques des déplacements possibles.

Une IHM sera présentée à l'utilisateur sur l'écran tactile¹¹ 4,3 pouces afin que celui-ci puisse tester de manière individuelle tous les éléments constitutifs de la tête.

Par exemple l'utilisateur demande à « scanner » le bus série TTL des servomoteurs Dynamixel pour déterminer ceux qui sont « actifs ». Puis à partir de la liste élaborée précédemment, l'utilisateur peut choisir un servomoteur. Les paramètres du servomoteur doivent alors s'afficher et pour certains permettre une modification en temps réel.

Pendant la phase de développement, l'écran pourra afficher des messages de déverminage.

Les IHM seront réalisées en utilisant la bibliothèque Qt¹² sous Linux pour la carte BeagleBoneBlack, et en utilisant les WindowsForms sous Windows grâce au projet Mono¹³ pour la carte RaspberryPi.

⁵<http://www.raspberrypi.org/> et <http://innovelectronique.fr/2013/06/09/raspberrypi-a-tout-faire/>

⁶<http://beagleboard.org/BLACK> et <http://elinux.org/Beagleboard:BeagleBoneBlack>

⁷<http://cmusphinx.sourceforge.net/> et <http://cmusphinx.sourceforge.net/2010/03/pocketsphinx-0-6-release/>

⁸<http://espeak.sourceforge.net/>

⁹<http://tcts.fpms.ac.be/synthesis/mbrola.html>

¹⁰<http://opencv.org/>

¹¹<http://www.lextronic.fr/P29777-module-shield-afficheur-couleur-tft-28.html>

¹²<http://qt-project.org/>

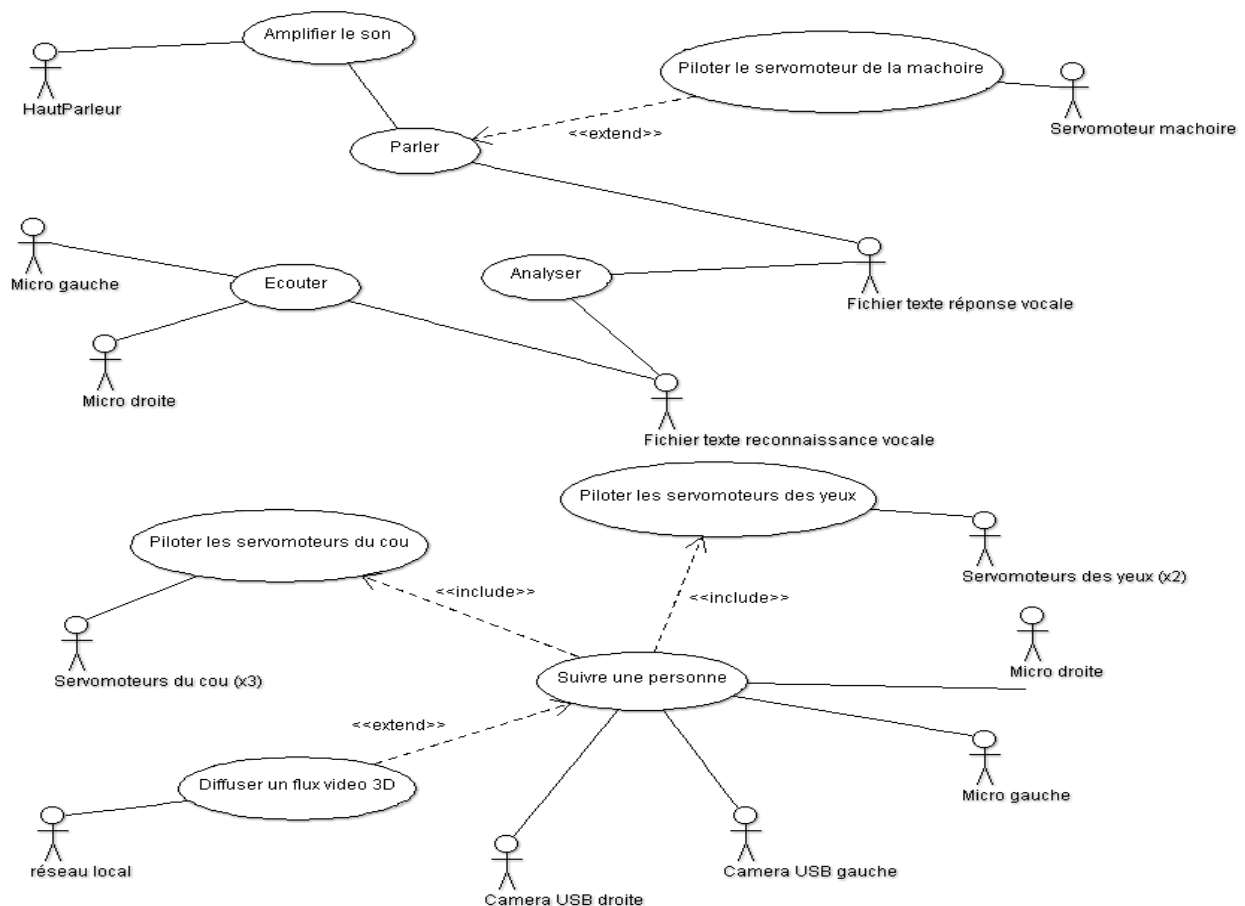
¹³<http://www.mono-project.com/>

2. Capture des besoins

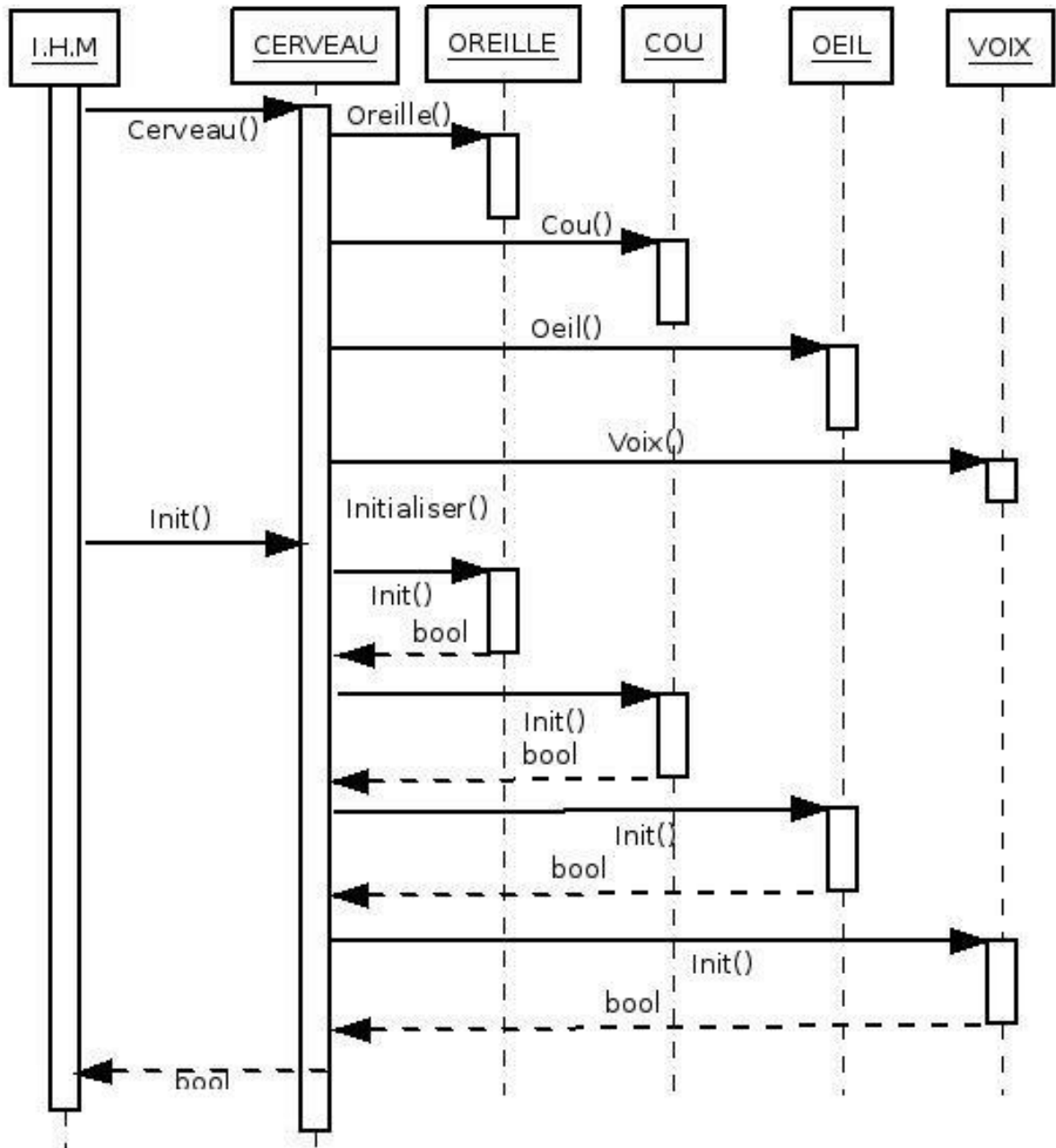
a. Contexte du système

Commanditaire	<i>Cybedroïd</i>
Projet nouveau	<i>oui</i>
Projet interne	<i>non</i>
Délais de réalisation	<i>Début : janvier 2015 Fin : mai 2015</i>
Investissement	<i>Montant : 1200€</i>
Équipe de développement	<i>8 étudiants</i>
Professeurs responsables	<i>Équipe STS IRIS</i>

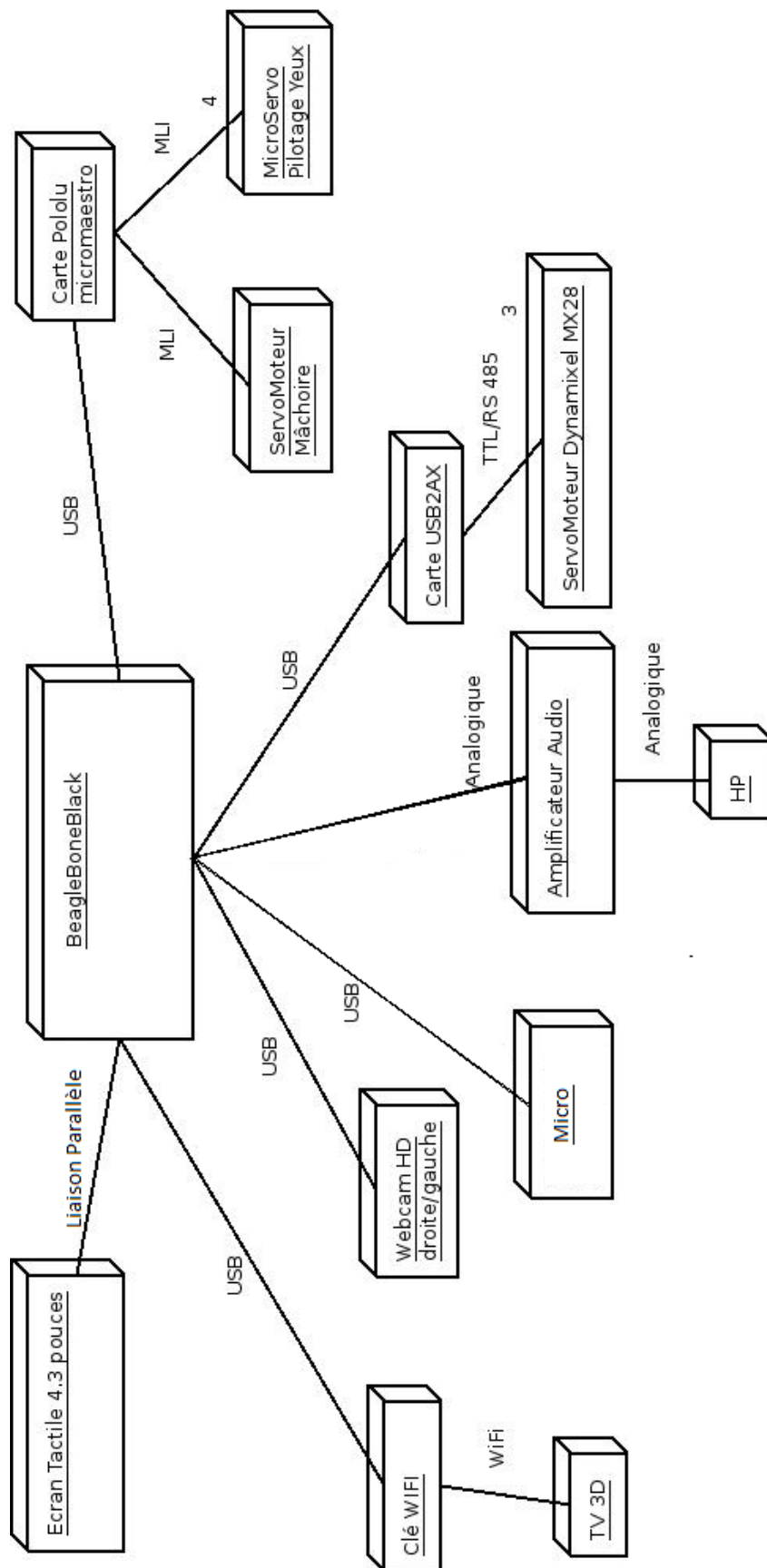
b. Ensemble des cas d'utilisations



c. Diagramme de séquence d'initialisation de la tête

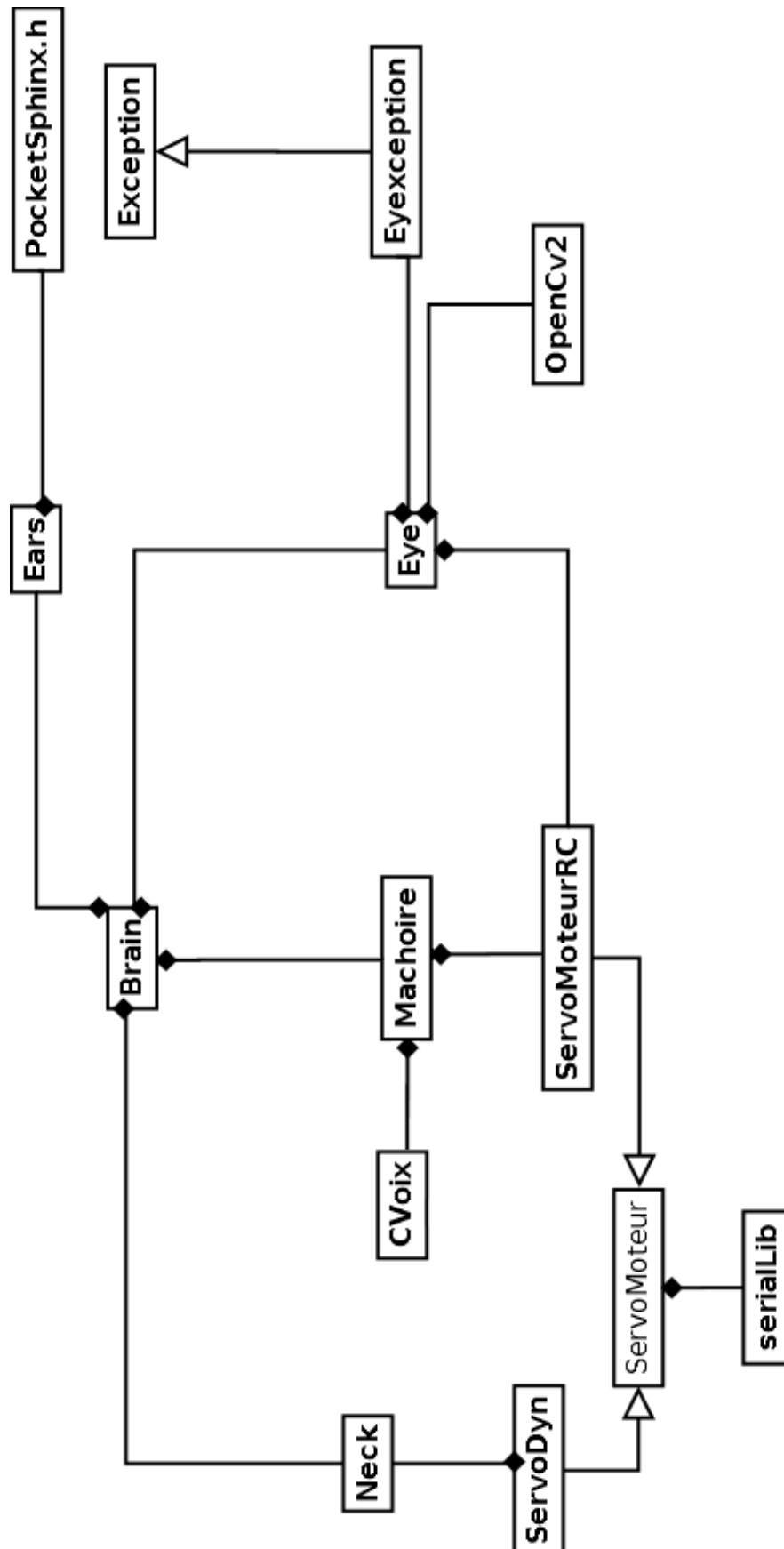


d. Diagramme de déploiement

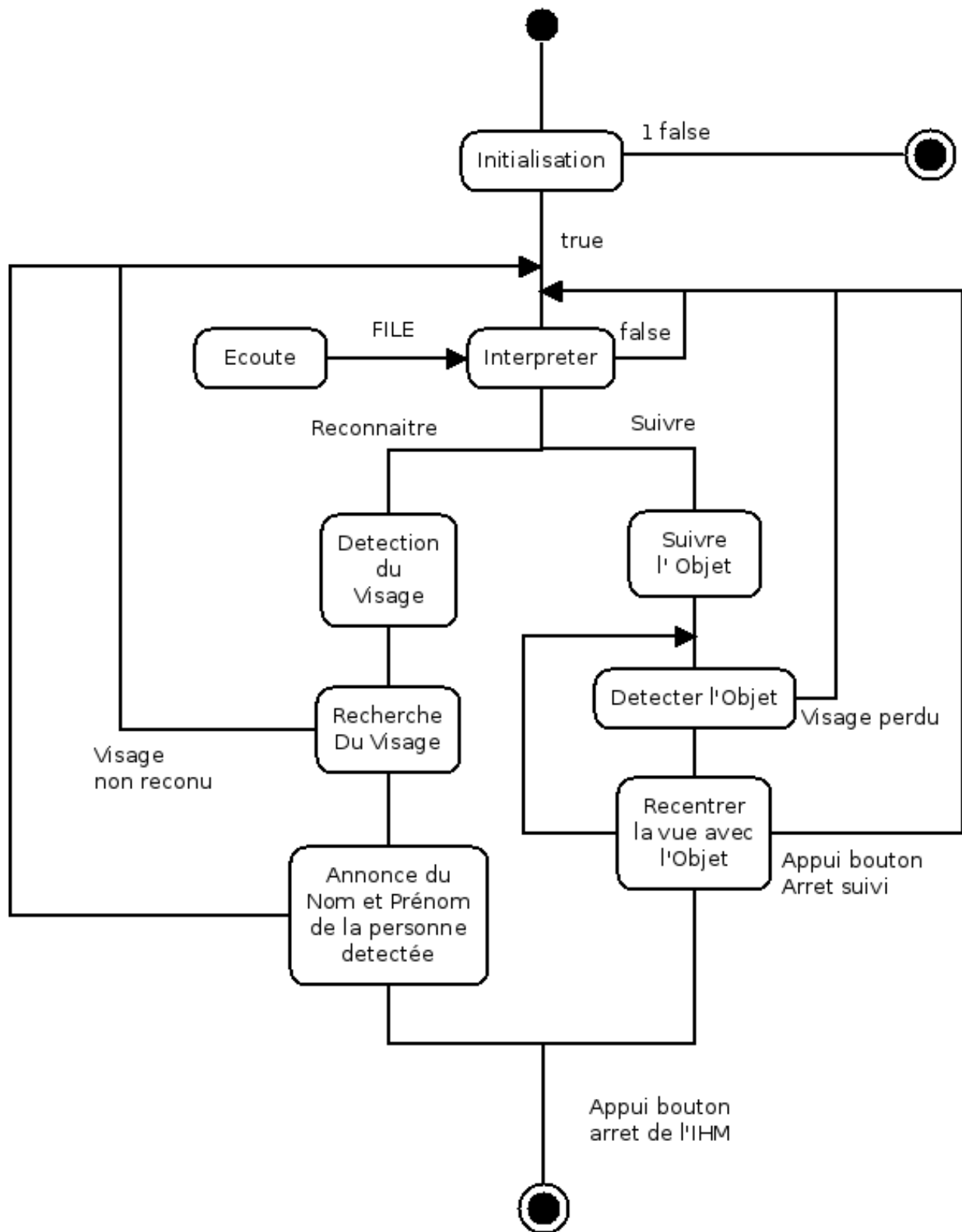


3. Analyse et conception préliminaire

a. Diagramme de classe



b. Diagramme Etat-transition



II. VEITES Thomas : Reconnaissance vocale et réalisation de l'I.H.M.

1. Rôle dans le projet

a. Reconnaissance Vocale

Afin de donner un effet humain à la tête robotisée, celle-ci doit intégrer un système de reconnaissance vocale, permettant ainsi à l'utilisateur d'interagir via sa voix sur la tête robotisée en prononçant des mots clés simples tel que « droite, gauche, haut, bas ».

b. Intégration des différentes Classes

Mise en relation des différentes classes développées par l'équipe. Intégration de celle-ci, afin d'obtenir un programme opérationnel ressemblant à une intelligence artificielle pouvant être contrôlée par la voix ou via une interface homme machine.

c. Interface Homme Machine

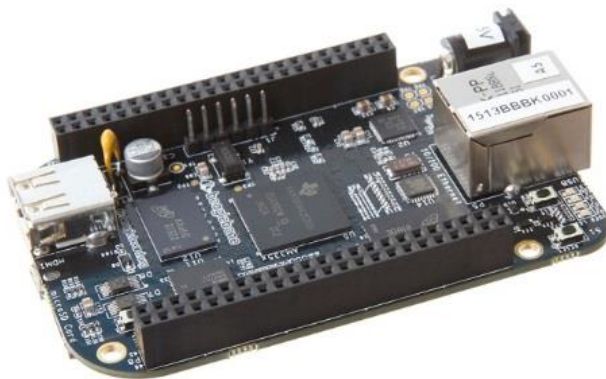
Afin de pouvoir contrôler la tête robotisée, sans la voix, je suis en charge du développement d'une I.H.M. (Interface Homme Machine) sur une carte BeagleBone Black permettant à l'utilisateur de prendre le contrôle sur les différentes parties indépendamment. Il sera possible de passer la tête en mode « automatique » via l'I.H.M. pour que celle-ci devienne autonome.

2. Mise en place

a. BeagleBone Black

Voici quelques caractéristiques de cette carte :

- ✚ CPU ARM Cortex- A8 cadencé 2GHz
- ✚ 512 Mo RAM DDR3
- ✚ 4Go eMMC
- ✚ Ethernet / 1 USB / MicroSD / Micro HDMI
- ✚ 65 GPIO (Entrée / Sortie - Tout ou Rien)
- ✚ 8 PWM
- ✚ 7 entrées analogiques
- ✚ 5 UART (ports série)
- ✚ 3 bus I2C
- ✚ 1 bus SPI
- ✚ 1 bus CAN



La BeagleBone Black peut démarrer un système d'exploitation à partir d'une carte SD ou à partir de l'eMMC, afin de gagner en rapidité, nous avons choisi d'installer notre système sur ce dernier.

b. Installation d'un SE sur BeagleBone

Il a fallu dans un premier temps installer un système d'exploitation sur la BeagleBone

Black, nous avons donc téléchargé l'image Debian 7.7 console armhf datant du 6 janvier 2015.

Ecriture de l'image en .img vers une carte SD via la commande suivante /dev/sdb représentant la carte SD :

```
dd if=debian7.7-console-armhf-2015-01-06.img of=/dev/sdb
```

Une fois l'image écrite sur la carte SD, il faut la copier sur la eMMC, pour cela la manipulation est assez simple : il faut insérer la carte SD dans la BeagleBone, appuyer sur le bouton « boot » (bouton à côté du port USB) puis mettre l'alimentation tout en maintenant le bouton jusqu'à ce que les 3 leds de la carte s'allument en mode « chargement » (les leds font comme un slide), après quelques minutes d'attente, nous pouvons débrancher la BeagleBone et retirer la carte SD.

Le système d'exploitation Debian 7.7 est installé sur l'eMMC.

c. Configuration réseau de la BeagleBone

N'ayant pas d'écran, la première configuration de la BeagleBone a été faite par la voie série grâce aux broches TX RX et un dongle USB simulant un port série.

Ainsi grâce au logiciel Putty (en mode voie série), nous nous sommes connectés à la carte avec les identifiants par défaut (debian debian).

Nous passons root via la commande :

```
sudo -i
```

Configuration du réseau via le fichier /etc/network/interfaces, ajout des lignes suivantes :

```
auto eth0
iface eth0 inet static
address 10.187.122.16
netmask 255.255.255.0
gateway 10.187.122.245
dns-nameserver 194.2.0.20
```

Afin de donner l'adresse IP 10.187.122.16 à notre BeagleBone.

Il faut redémarrer le service des connexions avec la commande suivante :

```
/etc/init.d/networking restart
```

Nous pouvons vérifier que nos paramètres du fichier /etc/network/interfaces ont été pris en compte en lançant un ping vers un site internet (Google par exemple) et voir si il répond.

d. Mise en place des utilisateurs

Nous avons créé un compte par utilisateurs, ainsi chacun a son répertoire personnel et peut travailler dans son dossier.

Un compte nommé auto a été ajouté, ce compte sera celui qui utilisera l'interface graphique.

Créer un compte utilisateur :

```
useradd -md /home/user user
passwd user
```

e. Installation d'un serveur SSH

Afin de permettre une connexion à distance des utilisateurs, nous avons installé un serveur SSH (Secure Shell) via la commande suivante :

```
aptitude install ssh
```

Les utilisateurs peuvent désormais se connecter sur la BeagleBone via leur compte depuis n'importe quelle machine connectée au réseau.

f. Installation interface graphique

Nous avons rajouté un écran tactile sur la carte BeagleBone, pour que l'utilisation des utilisateurs soit simple, il a fallu installer une interface graphique, nous avons choisi l'environnement XFCE 4 et le gestionnaire de session LightDM pour leur légèreté.

L'installation se fait via la commande suivante :

```
aptitude install xfce4 lightdm
```

Nous avons modifié le fichier de configuration `/etc/lightdm/lightdm.conf` grâce à la commande :

```
nano /etc/lightdm/lightdm.conf
```

Rajouter « auto » derrière l'option `autologin-user` :

```
autologin-user=auto
```

Ainsi au démarrage de la carte BeagleBone, la session auto est lancée automatiquement.

3. Reconnaissance Vocale

Pour la reconnaissance vocale, nous avons utilisé la bibliothèque PocketSphinx. PocketSphinx est aussi un programme open source développé en C.

a. A propos de PocketSphinx

PocketSphinx est une bibliothèque dépendant de SphinxBase. Basé sur une licence BSD, son codé peut être reproduit et utilisé aussi bien dans des programmes libres ou à but lucratif. La version utilisée dans notre système est la 0.8.

Il est possible d'installer PocketSphinx via les dépôts de la version instable de Debian (Sid) pour la BeagleBone mais nous avons opté pour une installation manuelle (téléchargement des packages).

b. Installation PocketSphinx

Avant d'installer PocketSphinx, il faut installer SphinxBase.

Les packages `pocketsphinx-0.8` et `sphinxbase-0.8` sont au format `.tar`, il suffit de les extraire via la commande :

```
tar -xvf monpackage.tar
```

L'installation de `sphinxbase` et de `pocketsphinx` est similaire, nous expliqueront donc une seule installation.

Une fois le package extrait, un dossier est créé, il suffit de rentrer dans ce dossier et de lancer le script `autogen.sh`, ce script crée un fichier de configuration en fonction de la machine utilisée. Nous pouvons ensuite lancer `configure` permettant de créer un `makefile` (fichier contenant des instructions de compilation).

Il suffit de lancer la commande « `make` » puis « `make install` » pour compiler et installer le tout dans le système.

Voici les commandes à utilisées chronologiquement :

```
cd sphinxbase
```

```
./autogen.sh
./configure
make
make install
cd ../pocketsphinx
./autogen.sh
./configure
make
make install
```

Cette installation permet d'utiliser le fichier « pocketsphinx.h » dans les programmes C++, mais installe aussi des exécutables tel que pocketsphinx_continuous, cet exécutable permet d'écouter et d'afficher les mots prononcés en permanence.

Ainsi, nous avons pu vérifier que l'installation a été réussie via la commande suivante :

```
pocketsphinx_continuous -adcdev plughw0,0
```

L'option adcdev permet de choisir la carte son sur la quel on va écouter.

c. Utilisation en C++

La bibliothèque PocketSphinx est codée en C.

Pour l'utiliser dans un projet C++, il faut inclure le fichier <pocketsphinx.h>. Il permet d'inclure des fonctions nécessaires à la conversion de la voix vers le texte. Trois fichiers sont obligatoires pour l'utilisation de pocketsphinx :

- LM : Language Model
- DIC : le Dictionnaire contenant les mots.
- HMM : Hidden Markov Model

Lest tests unitaires ont été fait avec les fichiers de la langue Anglaise.

Afin que la tête reconnaisse les mots de la langue de Molière, nous avons téléchargé un package comprenant le DIC, LM et HMM.

Pour prendre en compte ces nouveaux fichiers, il faut écrire la commande suivante :

```
pocketsphinx_continuous -adcdev plughw:1,0 \
-hmm /home/thomas/CMUS_FR/lium_french_f0 \
-dict /home/thomas/CMUS_FR/frenchWords62K.dic \
-lm /home/thomas/CMUS_FR/french3g62K.lm.dmp
```

d. Classe Ears

La classe Ears s'occupe de la conversion de la voix vers du texte, c'est l'oreille de notre tête robotisée.

Pour construire cette classe, nous avons étudié le fonctionnement de pocketsphinx_continuous, Ears a été construite sur la base de ces fonctions.

Voici le diagramme de classe de Ears :

Ears
- lmDir: string - dicDir: string - hmmDir: string - ps_decoder_t*: ps - cmd_ln_t*: config - FILE*: fh
- updateModel(): int + init(): int + listenMicro(): string + close(): void

Classe Ears

Explication des différentes fonctions :

- **init()** : Cette fonction configure les arguments et les chemins des fichiers de langue.
- **updateModel()** : Permet de recharger les fichiers de langues en cas de changement

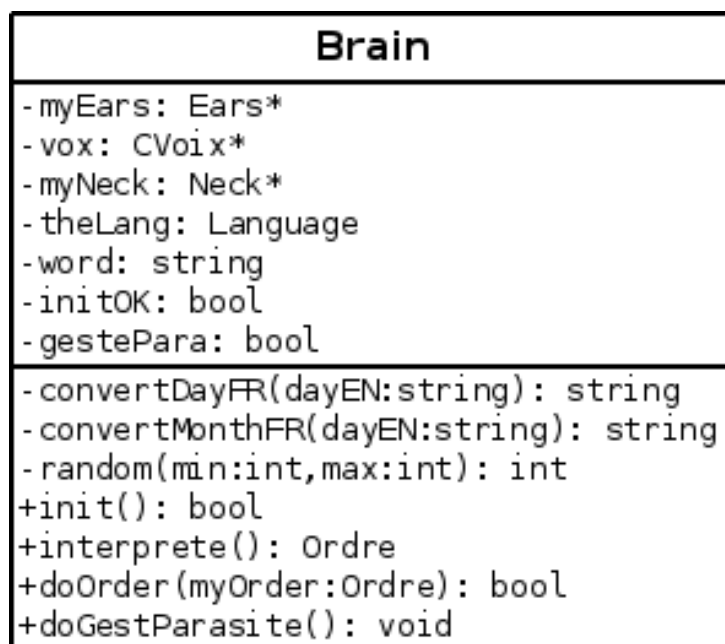
- via une fonction set.
- **listenMicro()** : Fonction bloquante, elle attend, quand on parle, elle déclenche une écoute et s'arrête lorsque la phrase est terminée. Elle renvoi le texte compris dans un string.
- **close()** : Cette fonction arrête proprement les objets nécessaire a la classe créé précédemment.

Explication des différents attributs :

- **string lmDir** : Contient le chemin du fichier LM utilisé.
- **string dicDir** : Contient le chemin du fichier DIC utilisé.
- **string hmmDir** : Contient le chemin du fichier HMM utilisé.
- **ps_decoder_t *ps** : Pointeur sur un objet défini dans pocketsphinx.h permettant de décoder un fichier son vers du texte.
- **cmd_in_t *config** : Pointeur sur un objet défini dans pocketsphinx.h permettant de stocker la configuration qui sera utilisée.
- Les autres attributs (comme **ad_rec_t *ad**, **adbuf**, etc...) sont des attributs nécessaire à pocketSphinx.

e. Classe Brain

La classe Brain est la classe mère de ce projet, l'objet Brain met en relation les différentes partie qui ont été développé dans l'équipe, à savoir Eye, CVoix, Neck. Brain s'occupe aussi de l'intelligence artificielle de la tête robotisée.



Classe Brain

Explication des différentes fonctions :

- **init()** : Initialise les différents objets qui lui sont associer (Ears, Eye, ...)
- **setLangue()** : Permet de choisi la langue a utilisée.
- **splitString** : Permet de découper une chaine de caractère (string) en vecteur de string.
- **interprete()** : Cette fonction récupère une chaine découpe la phrase grâce à la fonction **splitString()** recherche un mot clé, et renvoi un Ordre (énumération dans Brain).
- **doOrder** : Cette fonction prend pour paramètre un Ordre (énumération dans Brain)

- et exécute des actions en fonction de l'ordre passé.
- **convertDayFR()** : Converti un jour (lundi, mardi, ...) de l'Anglais vers le Français.
 - **convertMonthFR()** : Converti un mois (janvier, février, ...) de l'Anglais vers le Français.
 - **random()** : Génère un nombre aléatoire compris entre un minimum et un maximum
 - Les fonctions **listen()** et **play()** sont des fonctions permettant de debugger car elle sont simplement un appelle des fonction **listenMicro()** et **parler()** de Ears et CVoix().

Explication des différentes fonctions :

- **Ears *myEars** : Pointeur sur un objet de type Ears permettant l'écoute.
- **CVoix *vox** : Pointeur sur un objet de type CVoix (classe développée par Mr Nicolas Louis) permettant la conversion du texte vers la voix.
- **Eye *myEye** : Pointeur sur un objet de type Eye (classe développée par Mr Florent Chaumette) permettant l'utilisation des caméras HD.
- **Neck *myNeck** : Pointeur sur un objet de type Neck (classe développée par Mr Kevin Stœbner) permettant le contrôle des servomoteurs Dynamixel.
- **bool gestePara** : True faire les gestes parasites, False ne pas faire les gestes parasites.
- **bool initOK** : Passe a vrai si l'initialisation a été faite.
- **string word** : Contient la dernière chaîne entendue par Ears.
- **time_t timer** : Permet de gérer le temps pour récupérer la date et l'heure actuelle.

f. Relations entre les classes

Voici le diagramme des classes complet :

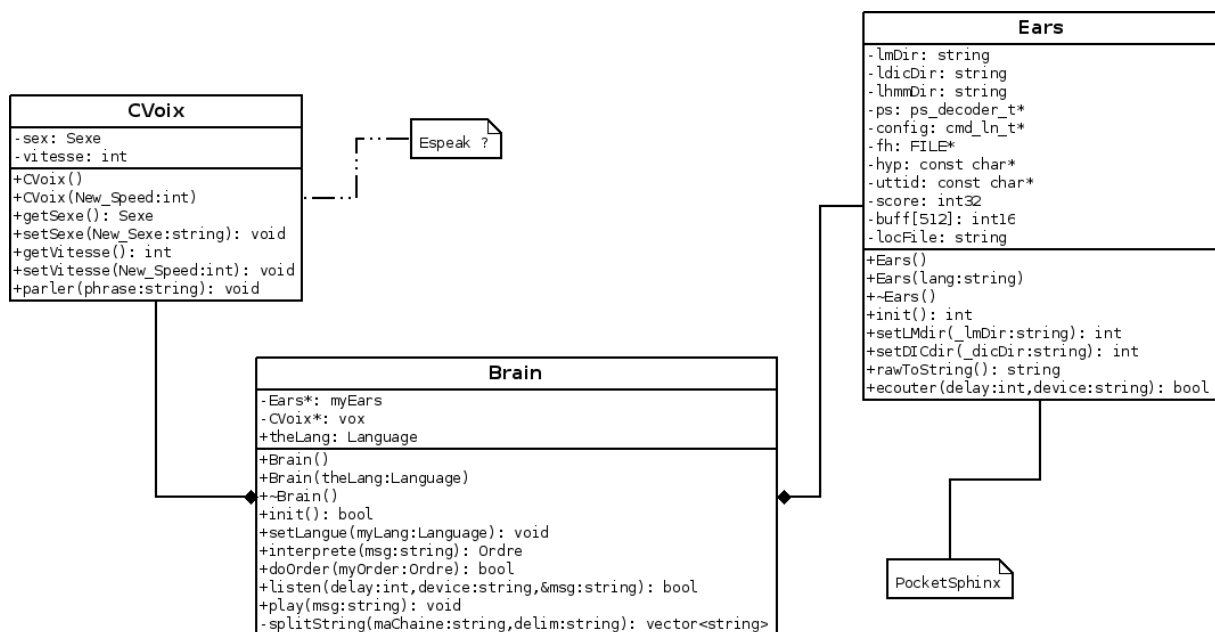
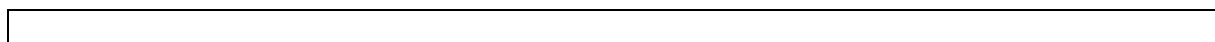


Diagramme global des Classes

Algorithme du fonctionnement dans le Main :



```
ordre : Order

Cerveau.init()

TANT QUE (ordre != Brain ::Ordre ::STOP)
    Cerveau.listen() ;
    Cerveau.interprete() → ordre
    Cerveau.doOrder(ordre)
FIN DU TANT QUE

RETURN 0
```

Nous pouvons remarquer que la classe Brain simplifie le code du programme principal.

g. Intelligence Artificiel

Afin d'éviter que la tête ne s'active sans demande de l'utilisateur, nous avons rajouté un système similaire au « OK Google », c'est-à-dire que pour demander un ordre à Josiane, il faut l'appeler par son prénom, une fois active vous pouvez demander un ordre.

Un ordre sera interprété si le mot qui correspond à celui-ci est contenu dans la phrase prononcée, ainsi l'utilisateur peut prononcer une phrase entière ce qui rend le système plus humain.

4. Le Framework Qt4

Qt est une découverte, il a fallu dans un premier temps se familiariser avec le programme afin de comprendre son fonctionnement. L'avantage majeur de Qt est qu'il est multiplateforme un projet développé sous Linux peut être repris et recompilé sous Windows ou Mac.

a. A propos de Qt4

Qt est une API (Application Programming Interface) orientée objet. Qt permet de concevoir une interface graphique en C++ via un système de Widget ou il suffit de glisser / déposer. L'environnement graphique KDE a utilisé la bibliothèque Qt. Ce qui a suscité de nombreuses critiques dans la communauté du logiciel libre car celle-ci était propriétaire. La société Trolltech fit passer les versions GNU/Linux et UNIX de Qt sous licence GNU GPL.

QtCreator génère un fichier nommé « nomClasse.ui » en langage XML.

Chaque Widget créé contient des événements appelés Signaux, lorsque un événement est exécuté, l'objet envoie un signal, par exemple un bouton qui sera cliqué renverra un signal « clicked ».

Il est possible de connecter un signal avec une action nommée Slot.

Nous connecterons donc des widgets (bouton, onglet, etc...) de l'I.H.M. aux actions de nos différentes classes.

b. Installation de Qt4

Nous avons installé le logiciel QtCreator sur un ordinateur, la BeagleBone n'étant pas pratique pour développer en graphique pour cause de son petit écran et de sa puissance. Le développement du programme c'est donc fait sur un ordinateur.

Pour installer Qt4 :

```
aptitude install qtcreator
```

Afin de ne pas installer toute l'interface de développement Qt4 sur BeagleBone, nous n'avons installé que les paquets nécessaires à la compilation de programmes via la commande :

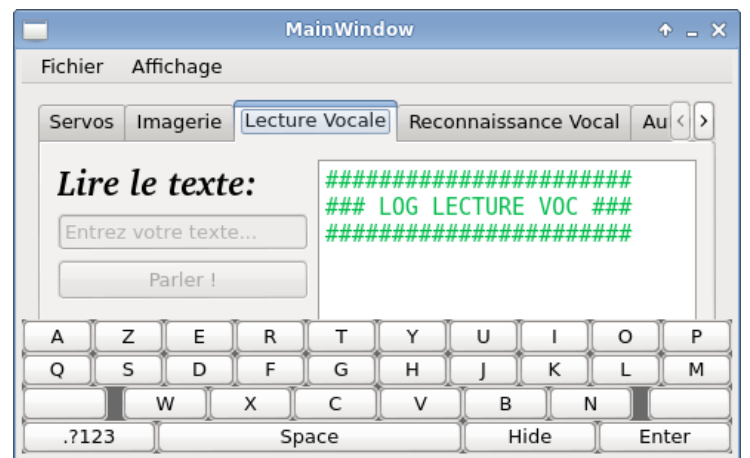
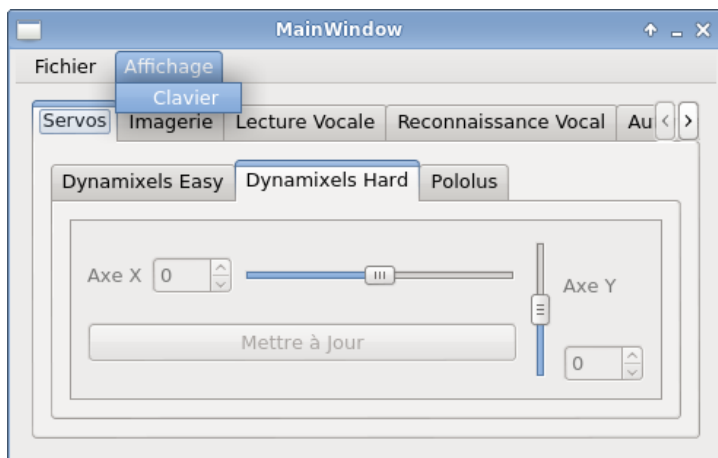
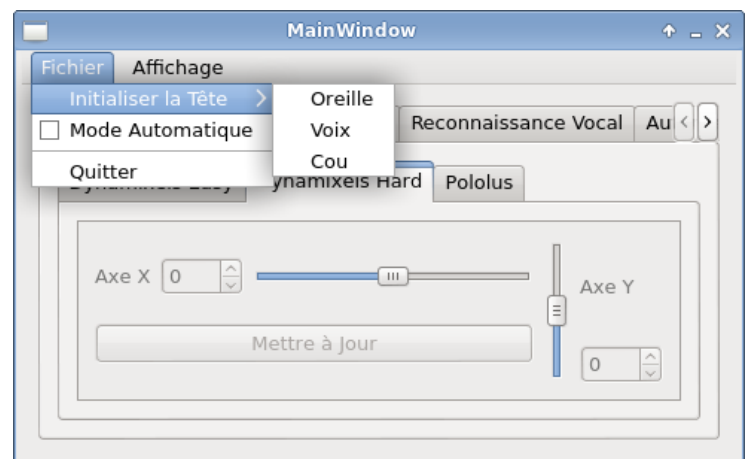
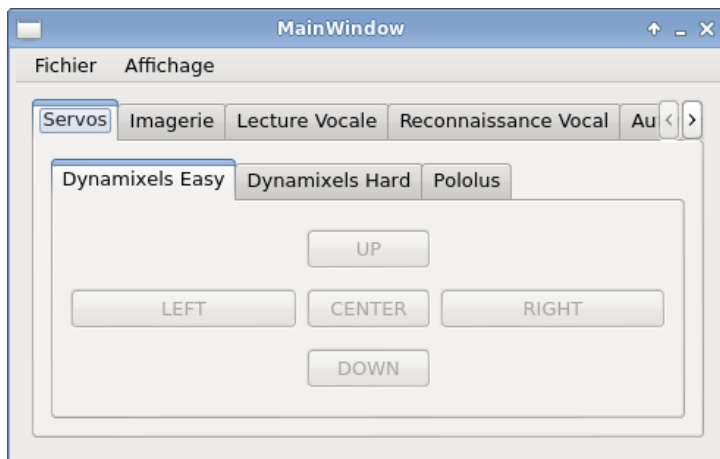
```
aptitude install libqt4-dev
```

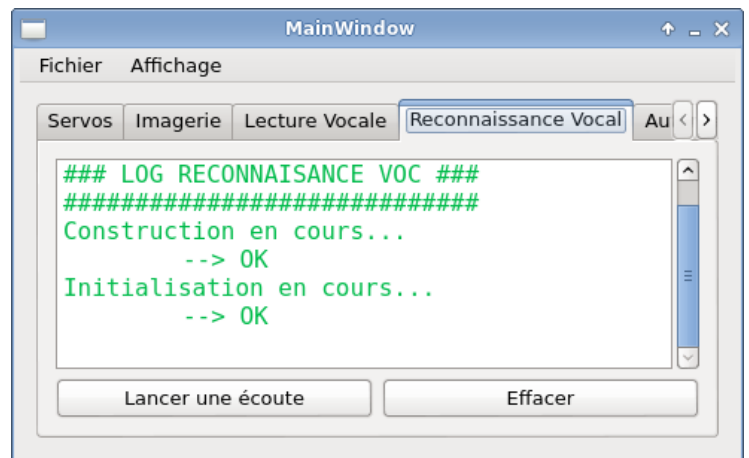
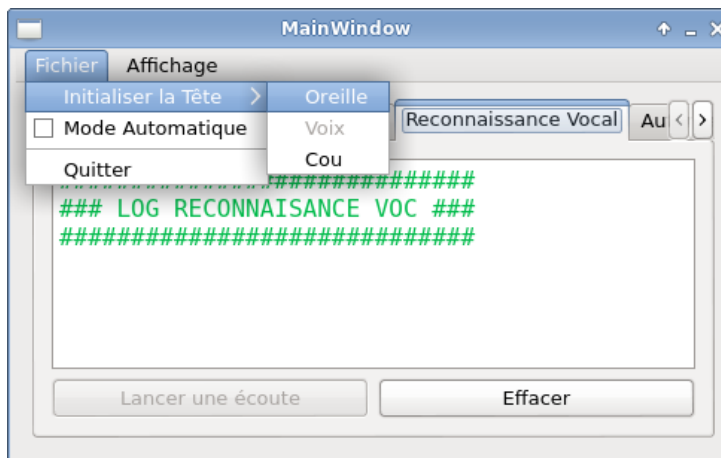
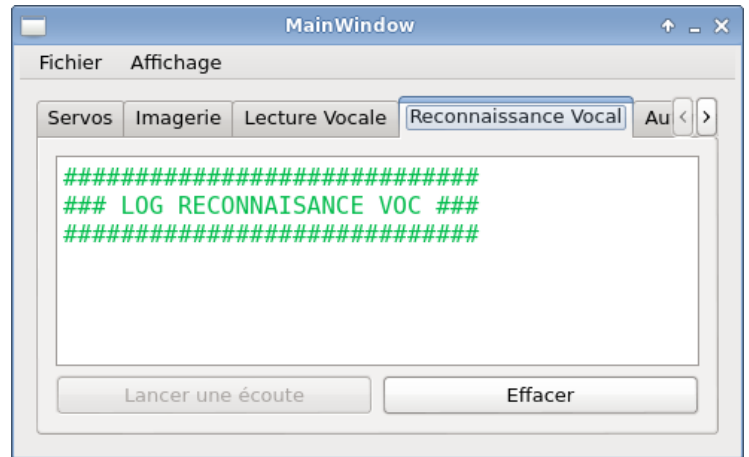
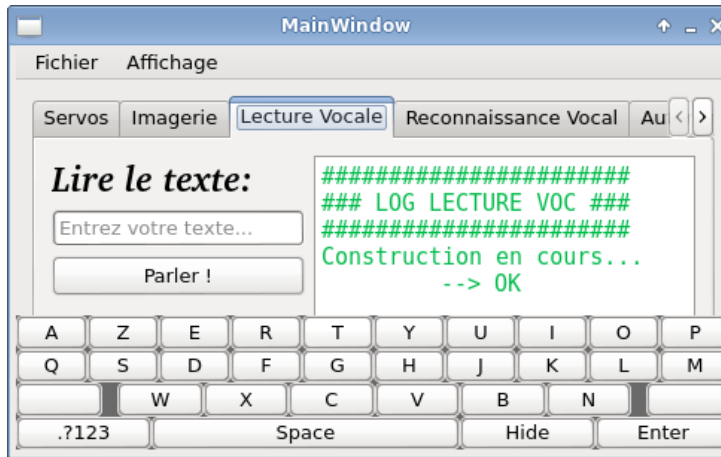
c. Présentation de L'Interface Homme Machine

L'Interface Homme Machine est composé d'un onglet par partie de la tête, soit :

- **Imagerie** : Permet d'afficher les images vues par les caméras HD intégrés à la tête.
- **Synthèse vocal** : Permet de convertir du texte en voix
- **Reconnaissance vocale** : Permet de convertir de la voix vers du texte
- **Servomoteur** : Permet de contrôler les servomoteurs soit avec un ordre prédéfinis (haut, bas, etc...) ou avec des valeurs X et Y.
- **Mode Automatique** : Permet de passer la tête en mode automatique pour que celle-ci devienne autonome

Quelques images de l'Interface Homme Machine réalisé sous Qt4 :

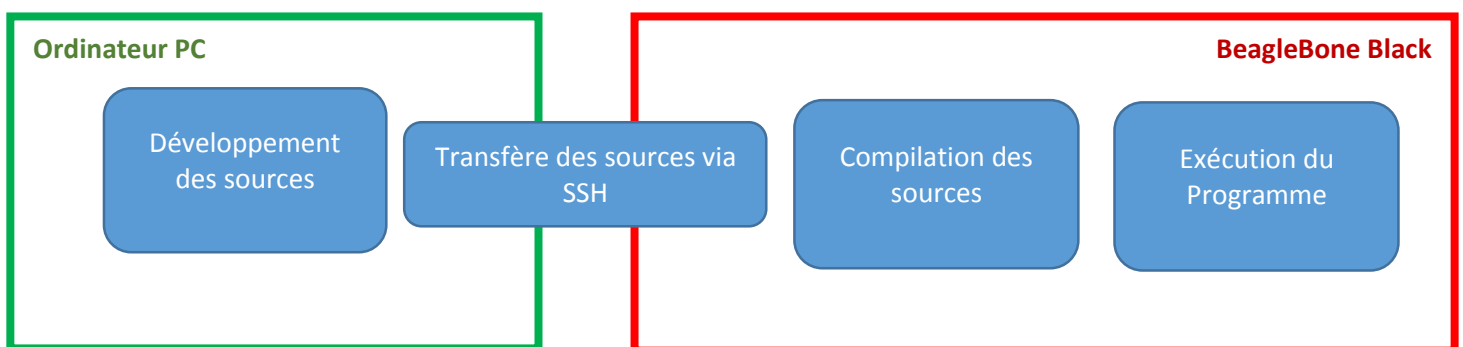




d. Compilation de l'IHM sur BeagleBone Black

Je n'ai pas utilisé la compilation croisé ni la compilation à distance car il m'a semblé difficile de la mettre en place avec le Framework Qt4.

Nous avons donc opté pour une compilation dite « native » :



Nous transférons les sources du PC vers la BeagleBone (IP : 10.187.122.16) via la commande suivante :

```
scp -r IHM/ thomas@10.187.122.16:/home/thomas/Developpement/
```

Nous nous connectons ensuite à la BeagleBone :

```
ssh thomas@10.187.122.16
```

Une fois connecté, nous pouvons nous rendre dans notre dossier transféré précédemment et lancer les commandes suivantes :

```
cd /home/thomas/Developpement/  
qmake-qt4  
make
```

- **qmake-qt4** : Génère un fichier « Makefile » contenant les instructions de compilation du programme.
- **make** : Exécute le Makefile et compile les sources afin d'obtenir un exécutable.

e. Problèmes rencontrés

Je vais vous présenter les différents problèmes qui ont été rencontré dans ma partie :

- **Problème de langue :**

Lorsque nous avons lancé nos tests unitaires de la classe Ears avec la langue Française, le temps de traitement a été extrêmement allongé (~3sec pour la langue Anglaise, et ~3min pour la langue Française).

Solution :

Nous avons pu remarquer que les fichiers de configuration de la langue Française étaient près de 10 fois plus lourds que les fichiers Anglais.

Nous avons donc recréé notre propre dictionnaire à partir de l'ancien via la commande suivante :

```
cat dico.dic.orig | grep monMot >> newDico.dic
```

On affiche le dictionnaire original, on récupère les mots qui nous intéresse et on les inscrits dans le nouveau dictionnaire.

- **Problème de puissance :**

Au fur et à mesure que le projet avançait, nous avons pu constater que la BeagleBone n'était pas adapté pour la tête par manque de puissance. En effet, simplement la reconnaissance vocale utilise près de 99% du processeur.

Solution :

Utiliser un ordinateur ou une carte plus puissante.

5. Conclusion

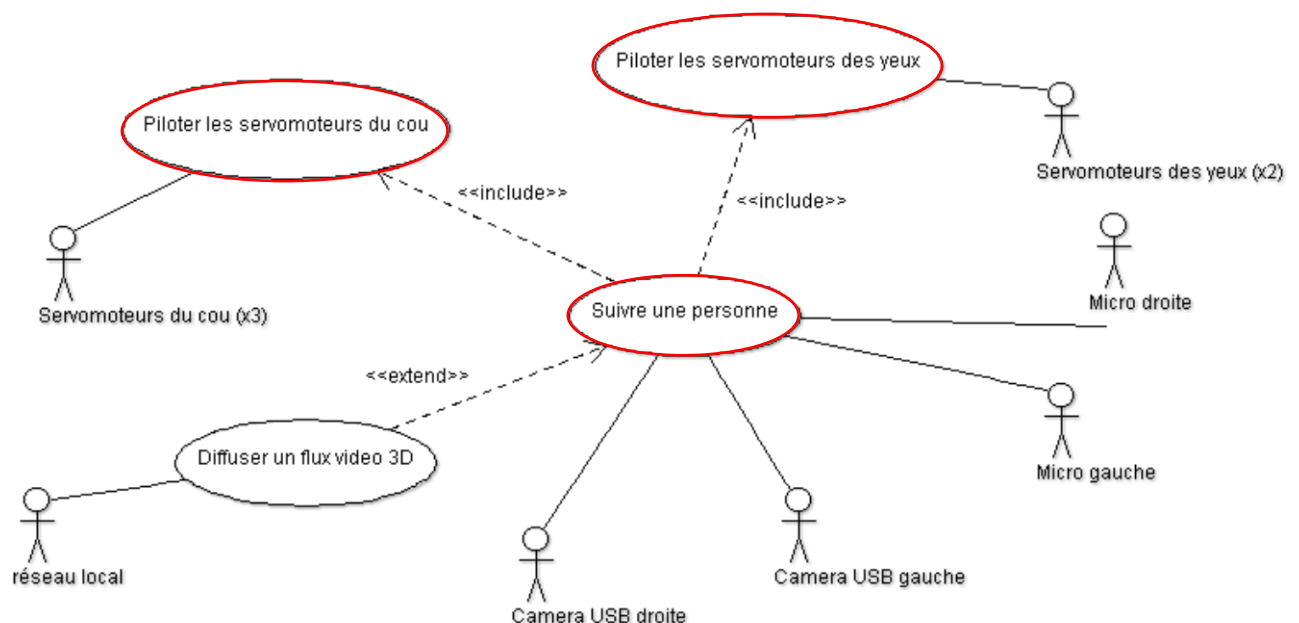
Ce projet n'est pas encore entièrement fonctionnel, mais nous avons bien avancé. Une carte plus puissante et une batterie aurai permis de rendre la tête réellement autonome. Nous avons pu remarquer que le développement d'une intelligence artificielle répondant dans toutes les situations est difficile et complexe.

Ce projet est tout de même bien avancer et la tête fonctionne avec un air humanoïde.

III. CHAUMETTE Florent : Suivi et reconnaissance faciale

1. Situation dans le projet

- Détecter les visages
- Reconnaître une personne
- Suivre une personne



Diagrammes des cas d'utilisation, les cas entourés sont ceux gérés dans cette partie.

2. OpenCv

a. Introduction

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage assure le support de cette bibliothèque depuis 2008. Cette bibliothèque est distribuée sous licence BSD. *(La licence BSD (Berkeley Software Distribution License) est une licence libre utilisée pour la distribution de logiciels. Elle permet de réutiliser tout ou une partie du logiciel sans restriction, qu'il soit intégré dans un logiciel libre ou propriétaire).*

L'utilisation de cette bibliothèque étant nouvelle, il a fallu au préalable effectuer des recherches. Pour cela, le site <http://docs.opencv.org/doc/tutorials/tutorials.html> (en anglais) propose de nombreux tutoriaux et explications pour la programmation avec OpenCv.

Cette bibliothèque permet entre autre :

- **Le traitement d'image** : propose la plupart des opérations classiques en traitement bas niveau des images.
- **Le traitement vidéo** : propose un nombre important d'outils issus de l'état de l'art en vision des ordinateurs.
- **Utiliser des algorithmes d'apprentissage** : algorithmes classiques dans le domaine de l'apprentissage artificiel.

- **Des calculs matriciels** : une image peut être considérée comme une matrice de pixel. Ainsi, toutes les opérations de bases des matrices sont disponibles.
- **Et quelques autres fonctionnalités** : met également à disposition quelques fonctions d'interfaces graphiques, comme les curseurs à glissière, les contrôles associés aux événements souris, ou bien l'incrustation de texte dans une image.

b. Installation sur PC GNU/Linux Debian 7

Afin de comprendre le fonctionnement de la bibliothèque et de la prendre en main, il a fallu, dans un premier temps, l'installer sur un PC sous GNU/Linux Debian 7 (le plus proche de la configuration de la carte BeagleBone). Pour ce faire, il faut, en console, rentrer la commande suivante :

```
iris@iris:~$ sudo aptitude install libopencv-dev
```

Une fois les paquets installés, il faut paramétrer NetBeans, et donc le projet. Pour ce faire :

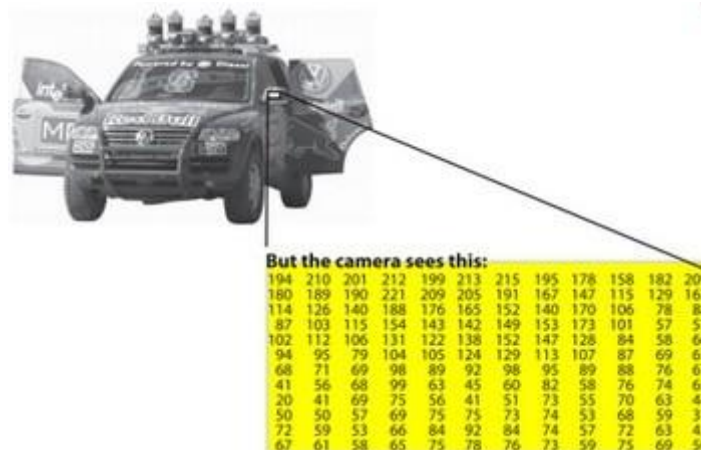
- clic-droit sur le projet>Propriétés>C++ Compiler>Include Directories & Headers>Add
- Chercher le dossier opencv2 dans /usr/include
- Dans Linker > Librairies > Add Librairies
- Chercher tout fichiers commençant par libopencv dans /usr/lib/x86_64-linux-gnu¹⁴

c. Prise en main d'OpenCv

L'ensemble des tests unitaires ont été réalisés sur un PC sous GNU/Linux.

Pour commencer, l'opération la plus simple est l'affichage d'une image.

L'utilisation de la bibliothèque OpenCv repose essentiellement sur la manipulation de variables de type `cv::Mat`. Ce sont des objets représentant une matrice. En effet, les images capturées par n'importe quelle caméra sont stockées dans ces dernières sous forme de matrice.



En haut, ce qu'une personne voit. En bas, ce que la caméra stocke.

Ces objets peuvent donc stocker des images que l'on pourra ensuite afficher. Pour stocker une image, il faut au préalable créer un objet `cv::Mat` puis utiliser la fonction :

```
cv::Mat cv::imread(const string& filename, int flags)
```

- filename, chemin vers l'image à stocker
- flags, option d'ouverture d'image (l'image peut être stockée en niveau de gris, canaux RGB (rouge / vert / bleu) inversés, etc... Dans notre cas, l'image sera chargée, la plupart du temps, sans options)

Une fois l'image chargée, on peut l'afficher. Pour cela, on utilisera la fonction :

```
cv::imshow(const string& winname, InputArray mat)
```

¹⁴ Pour les versions antérieures à 2.4, les libopencv se trouvent dans /usr/lib/

- winname, titre de la fenêtre affichée (si la fonction est appelée plusieurs fois avec le même titre de fenêtre, la précédente fenêtre sera remplacée par la nouvelle. Si la fonction est appelée plusieurs fois avec des titres différents, aucune fenêtre ne sera fermée et donc elles seront toutes affichées sur l'écran)
- mat, l'objet à afficher. Dans notre cas, nous utiliserons les cv::Mat.

Soit l'algorithme suivant :

Entrée : Chemin vers l'image à afficher

Variables : image cv::Mat

Début

Initialisation d'image, création de l'objet cv::Mat

Appel de la fonction cv::imread pour charger l'image passée en paramètre dans la variable image

Appel de la fonction cv::imshow pour afficher image dans une fenêtre

Appel de cv::waitKey pour attendre l'appui d'une touche

Fin

Enfin, ce programme de test unitaire permet d'afficher une image passée en paramètre :

```
#include <cstdlib>
#include <opencv2/opencv.hpp>

int main(int argc, char** argv) {
    cv::Mat image;
    image = cv::imread(argv[1]);
    cv::imshow("Titre fenêtre", image);
    cv::waitKey();
    return 0;
}
```

La fonction cv::waitKey(int delay) est bloquant (pendant un certain temps, le nombre de secondes passées en paramètres. S'il n'y a pas de paramètres, ou que celui-ci vaut 0 ou -1, le temps d'attente est infini) et permet d'attendre l'appui d'une touche avant de continuer le programme.

On lance le programme via une ligne de commande en indiquant le chemin vers l'image à afficher :

```
root@debian:~/Documents/Josiane_Flo/OpenCV/Test/Unitaire/AffichageImage/dist/Debug/GNU-Linux-x86# ./affichageimage /root/Documents/Josiane_Flo/Images/Lena.jpeg
Ici, on affichera l'image se trouvant dans /root/Documents/Josiane_Flo/Images/Lena.jpeg
```

D'où le résultat :



Cette image est très utilisée dans le traitement numérique d'image

OpenCv peut aussi récupérer des images via un flux vidéo. En effet, grâce à une webcam, il est possible de récupérer des images. Il est aussi possible de recréer un flux vidéo en affichant toutes les images captées par la webcam. Nous verrons d'abord la récupération d'image, puis ensuite, la création du flux vidéo.

(http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html)

Ici, la caméra utilisée est la caméra de la tête du robot, une Hercules Twist HD. La ligne de commande dmesg après la connexion de la caméra permet de vérifier si elle est reconnue et d'obtenir son id.

```
root@debian:~# dmesg
[12510.124387] usb 2-2: new full-speed USB device number 3 using ohci-pci
[12510.391997] usb 2-2: New USB device found, idVendor=80ee, idProduct=0030
[12510.392006] usb 2-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[12510.392008] usb 2-2: Product: VirtualBox Webcam - Hercules HD Twist
[12510.392591] usb 2-2: Manufacturer: VirtualBox
[12510.392598] usb 2-2: SerialNumber: 479d05e2a39e901b
[12510.432942] media: Linux media interface: v0.10
[12510.440930] Linux video capture interface: v2.00
[12510.468745] uvcvideo: Found UVC 1.00 device VirtualBox Webcam - Hercules HD Twist (80ee:0030)
[12510.486186] input: VirtualBox Webcam - Hercules HD as /devices/pci0000:00/0000:00:00:06.0/usb2/2-2/2-2:1.0/input/input8
[12510.486592] usbcore: registered new interface driver uvcvideo
[12510.486596] USB Video Class driver (1.1.1)
```

Ici, la caméra est reconnue, et a comme ID 3.

Premièrement, nous aurons besoin d'un nouvel objet, la VideoCapture. Cet objet permettra certaines opérations sur le flux vidéo. Il faut donc initialiser un objet VideoCapture en passant comme paramètre le chemin vers la webcam ou l'id de celle-ci (un paramètre valant 0 ouvre la première caméra détectée).

Ensuite la fonction VideoCapture::open() permettra d'ouvrir le flux vidéo (inutile si l'objet VideoCapture a un paramètre lors de sa création). Cette méthode renvoie un booléen pour vérifier si le flux est bien ouvert.

Enfin, pour extraire une image de ce flux, il faut utiliser la méthode VideoCapture::read(Mat& image) qui chargera la dernière image prise dans la variable image. Il suffira ensuite d'afficher image grâce à la fonction cv::imshow().

Pour créer un flux vidéo, il suffit de reprendre l’affichage d’une seule image issue de la caméra et de répéter cette action.

Ce qui donne en algorithme :

Entrées : (optionnel) chemin vers la caméra

Variables : image cv::Mat, camera cv::VideoCapture

Début

Création de cv::Mat image et cv::VideoCapture camera et initialisation

Appel méthode VideoCapture::open()

image <- Appel méthode VideoCapture::read()

Appel de la fonction cv::imshow pour afficher image

Appel cv::waitKey() pour attendre l'appui d'une touche

Fin

Algorithme pour afficher une image du flux vidéo

Entrées : (optionnel) chemin vers la caméra

Variables : image cv::Mat, camera cv::VideoCapture

Début

Création de cv::Mat image et cv::VideoCapture camera et initialisation

Appel méthode VideoCapture::open()

Tant que Condition de sortie

 image <- Appel méthode VideoCapture::read()

 Appel de la fonction cv::imshow pour afficher image

Fin Tant que

Appel cv::waitKey() pour attendre l'appui d'une touche

Fin

Algorithme pour afficher un flux vidéo via un autre

D'où les codes correspondants :

```
#include <cstdlib>
#include <opencv2/opencv.hpp>

int main() {
    cv::Mat image;
    cv::VideoCapture camera(0);
    camera.read(image);
    cv::imshow("Titre fenêtre", image);
    cv::waitKey();
    return 0;
}
```

Affiche une seule image issue de la caméra

```
#include <cstdlib>
#include <opencv2/opencv.hpp>

int main() {
    cv::Mat image;
    cv::VideoCapture camera(0);
    while (true) {
        camera.read(image);
        cv::imshow("Titre fenêtre", image);
        cv::waitKey(1);
    }
    cv::waitKey();
    return 0;
}
```

Affiche un flux vidéo issu de la caméra¹⁵

3. Cas d'utilisation « Détecter les visages »

a. Capture du besoin

Afin de reconnaître des visages, il faut au préalable détecter des visages. La bibliothèque OpenCv propose une méthode afin de détecter les éléments du visage que l'on souhaite.

b. Analyse et Conception préliminaire

La bibliothèque OpenCv propose des méthodes de détection de visages à l'aide d'un fichier xml, appelé haarCascade, qui est caractérisé par ce que l'on recherche. Par exemple, un fichier xml détectera les yeux sur une image alors qu'un autre détectera le visage entier. Dans notre partie, on utilisera deux fichiers xml, un pour la détection de visages et un pour la détection des yeux (pour une détection plus fiable)

Les haarCascade sont composés de nombreuses balises et caractéristiques « pseudo-haar » ou « haar-like ». Ces caractéristiques sont utilisées en vision par ordinateur pour la détection d'objet dans des images numériques. Très simples et très rapides à calculer, elles ont été utilisées dans le premier détecteur de visages en temps réel, celui de la méthode de Viola et Jones.¹⁶

¹⁵ L'appel de la fonction cv::waitKey(1) entouré en rouge est obligatoire sinon les images ne s'affichent pas.

¹⁶ Source et compléments : http://fr.wikipedia.org/wiki/Caract%C3%A9ristiques_pseudo-Haar

```

- <!-- node 1 -->
▼<feature>
  ▼<rects>
    <_>12 12 6 4 -1.</_>
    <_>12 14 6 2 2.</_>
  </rects>
  <tilted>0</tilted>
</feature>
<threshold>7.6601309701800346e-003</threshold>
<left_val>0.5411052107810974</left_val>
<right_val>0.2180687040090561</right_val>
</_>
</_>
▼<_>
  <!-- tree 8 -->
  ▼<_>
    <!-- root node -->
    ▼<feature>
      ▼<rects>
        <_>1 11 6 3 -1.</_>
        <_>1 12 6 1 3.</_>
      </rects>
      <tilted>0</tilted>
    </feature>
    <threshold>7.6467678882181644e-003</threshold>
    <left_node>1</left_node>
    <right_val>0.1158960014581680</right_val>
  </_>
</_>

```

Extrait du haarCascade de détection de visage

c. Mise en place de l'environnement du système

Pour utiliser ces fichiers, nous allons créer un nouvel objet : le `cv::cascadeClassifier` auquel on passera en paramètre le chemin vers le fichier `haarCascade`.

Une fois initialisé, l'objet `cv::cascadeClassifier` propose une méthode

`cascadeClassifier::detectMultiScale(Mat img, std::vector<cv::Rect> tabRect)`

- `img`, l'image sur laquelle les visages seront détectés.
- `tabRect`, un vecteur d'objet `cv::Rect` qui récupèrera les rectangles situés autour des visages que la fonction générera.

La méthode `cascadeClassifier::detectMultiScale()` transforme l'image passée en paramètres en niveau de gris afin de travailler sur les zones contrastées. En effet, la zone autour des yeux d'un visage a un contraste plus important que n'importe quelle autre zone. L'arête du nez sépare cette zone en deux parties qui seront identifiées comme des yeux, puis le visage est détecté autour de cette zone.

Soit en algorithme :

Entrées : image, `cv::Mat` - `tabRect`, `std::vector<cv::Rect>`

Variables : `cascade`, `cv::CascadeClassifier`

Début

Chargement de image

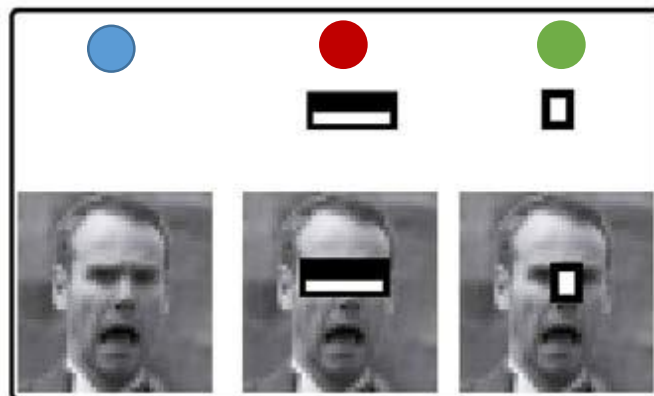
- Conversion en niveau de gris de image
- Parcours de l'image pour rechercher la partie la plus contrastée avec en dessous un contraste moins important
- Recherche d'une zone moins contrastée vers le milieu de la zone contrastée

Création d'un objet `cv::Rect` autour de la zone la plus contrastée

Ajout de l'objet `cv::Rect` dans `tabRect`

Fin

Algorithme de détection de visages



Fonctionnement de la méthode `detectMultiScale()`

Une fois la détection faite, il faut afficher les rectangles contenus dans le vecteur de `cv::Rect`. Pour ce faire, OpenCv propose une fonction :

`cv::rectangle(Mat& image, Rect rec, const Scalar color, int thickness, int lineType, int shift)`

- ✚ image, l'image sur laquelle on affichera les rectangles
- ✚ rec, Rect à afficher sur image
- ✚ color, type Scalar correspondant à la couleur du rectangle à afficher (Scalar color(NiveauRouge, NiveauVert, NiveauBleu))
- ✚ thickness, épaisseur du trait du rectangle
- ✚ lineType, non utilisé
- ✚ shift, non utilisé

Soit en algorithme :

Entrées :

Variables : cascade, `cv::cascadeClassifier` - image, `cv::Mat`
tabRect, `std::vector<cv::Mat>`

Debut

Appel de la méthode `cascade.detectMultiScale()`

Pour i allant de 0 à (taille de tabRect) Faire

Appel de `cv::rectangle(image, tabRect[i], Scalar(255,0,0), 2)`

Fin Pour

Appel de la fonction `cv::imshow()` pour afficher image

Fin

Algorithme d'affichage de rectangles

D'où le code suivant :

```
//Fichier de détection des visages
CascadeClassifier classif("/root/Documents/Josiane_Flo/Ressources/haarcascade_frontalface_alt2.xml");

//Variable contenant l'image
Mat image;
image = imread(argv[1]);
imshow("Image avant traitement", image);

//Attend l'appel d'une touche du clavier
waitKey(0);

//Vecteur contenant les rectangles qui seront autour du (des) visage(s) détectés
vector<Rect> tabRectFaces;

//Détection des visages grâce au fichier .xml
classif.detectMultiScale(image, tabRectFaces);

// Affichage des rectangles autour du (des) visage(s) détecté(s)
Scalar<int> color(255, 0, 0);
for (int i = 0; i < tabRectFaces.size(); i++) {
    rectangle(image, tabRectFaces.at(i), color);
}

//Affichage de l'image avec le(s) rectangle(s)
imshow("Image après traitement", image);
waitKey(0);
```

Le code ci-dessus affiche l'image passée en paramètre avant de détecter les visages et afficher l'image à nouveau avec les rectangles.

La détection est aussi possible sur un flux vidéo grâce à l'algorithme suivant :

Entrées :

Variables : cascade, cv::cascadeClassifier - camera, VideoCapture -
image, Mat - tabRect, std::vector<cv::Mat>

Début

Création et initialisation de camera, cascade et image

Tant que (condition de sortie)

 image <- camera.read()

 cascade.detectMultiScale(image, tabRect)

 Pour i allant de 0 à (taille de tabRect) Faire

 cv::rectangle(image, tabRect[i], Scalar(255,0,0), 2)

 Fin Pour

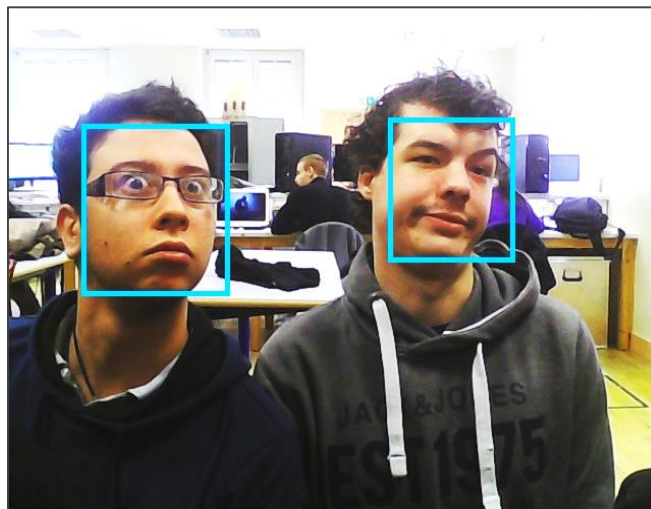
 Appel de cv::imshow pour afficher image

Fin Tant que

Fin

Algorithme de détections de visages sur flux vidéo

Cependant, cet algorithme mobilise beaucoup de ressources, pour parer cela, une solution est de mettre la détection (méthode detectMultiScale()) dans un thread à part).



Résultat de détection des visages sur une image

d. Conclusion sur l'avancement du travail

La bibliothèque est utilisable et fonctionnelle. Les images peuvent être affichées et les visages sont détectés. Les rectangles sont affichés même si plusieurs visages sont présents sur l'image. Enfin, les visages sont détectés sur un flux vidéo.

4. Cas d'utilisation : « Suivre une personne »

a. Capture des besoins

Lorsqu'une personne se présente devant les caméras, ou lorsqu'elle dit un mot clé, la tête doit suivre cette personne.

b. Analyse et Conception préliminaire

Après avoir détecté une personne grâce à la méthode de la bibliothèque OpenCv vue précédemment, on peut récupérer la position du rectangle se trouvant autour du visage de la personne, donc il est possible, grâce aux servomoteurs du cou et des yeux, de pouvoir recentrer ce rectangle sur l'écran donnant l'impression que la tête suit la personne.

c. Mise en place de l'environnement du système

Pour savoir dans quel sens tourner la tête, j'ai conçu une méthode (de la classe Eye, voir 6.) dont le prototype est int center(Mat img).

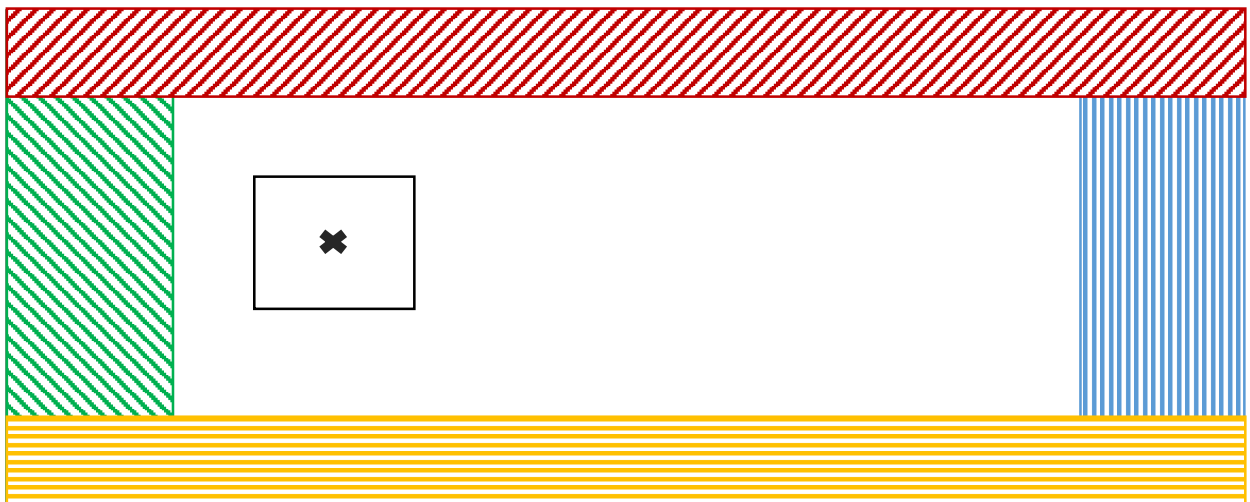
```
int Eye::center(Mat imageCenter) {
    detectMultiScale(image, Eye::flags::CASCADEHEAD);
    getCenter();
    int code = 0;
    if (posCenterImg.x < (30 / 100) * image.cols)
        code = code | 1;
    if (posCenterImg.x > (70 / 100) * image.cols)
        code = code | 2;
    cout << "Func : " << image.cols << endl;
    if (posCenterImg.y < (30 / 100) * image.rows)
        code = code | 4;
    if (posCenterImg.y > (70 / 100) * image.rows)
        code = code | 8;
    cout << "Func : " << image.rows << endl;
    return code;
}

void Eye::getCenter() {
    posCenterImg = Point(tabFaces.at(0).x + tabFaces.at(0).width / 2, tabFaces.at(0).y + tabFaces.at(0).height / 2);
}
```





Définition des méthodes center et getCenter de la classe Eye

La méthode center détecte les visages grâce à la méthode detectMultiScale() de la classe Eye (reprend la méthode cascadeClassifier::detectMultiScale()) puis appelle la méthode getCenter qui calcule le milieu du carré du premier visage détecté et l'envoi dans une variable Point de la classe.

La variable code se calcule de cette façon :



✖ Si est le centre du visage et qu'il se trouve dans une des parties hachurée on ajoute à la variable code :

-  - 1 si l'abscisse du centre se trouve dans la partie hachurée verte (< 30% de la largeur de l'image dans lequel le visage se trouve)
-  - 2 si l'abscisse du centre se trouve dans la partie hachurée bleue (> 70 % de la largeur de l'image dans lequel le visage se trouve)
-  - 4 si l'ordonnée du centre se trouve dans la partie hachurée rouge (< 30% de la hauteur de l'image dans lequel le visage se trouve)
-  - 8 si l'ordonnée du centre se trouve dans la partie hachurée orange (> 70 % de la hauteur de l'image dans lequel le visage se trouve)

Remarque : Si le centre du visage se trouve dans deux zones hachurées, les deux valeurs correspondantes seront additionnées. (Ex : Si le centre se trouve dans la zone rouge ET dans la zone verte (la zone verte ayant la même hauteur que l'image) la variable code recevra 1 + 4 donc 5)

La méthode recevant la variable code devra donc utiliser un masque pour savoir quel(s) bit(s) sont à 1 sur code et donc savoir quelle correction apporter.

d. Conclusion sur l'avancement du travail

Les servomoteurs du cou sont seront utilisés par la classe « Brain » et la classe des servomoteurs des yeux n'est pas complète.

5. Cas d'utilisation : « Reconnaître une personne »

a. Capture des besoins

Reconnaître une personne passant devant la tête, et, optionnellement, avoir une salutation personnalisée.

b. Analyse et Conception préliminaire

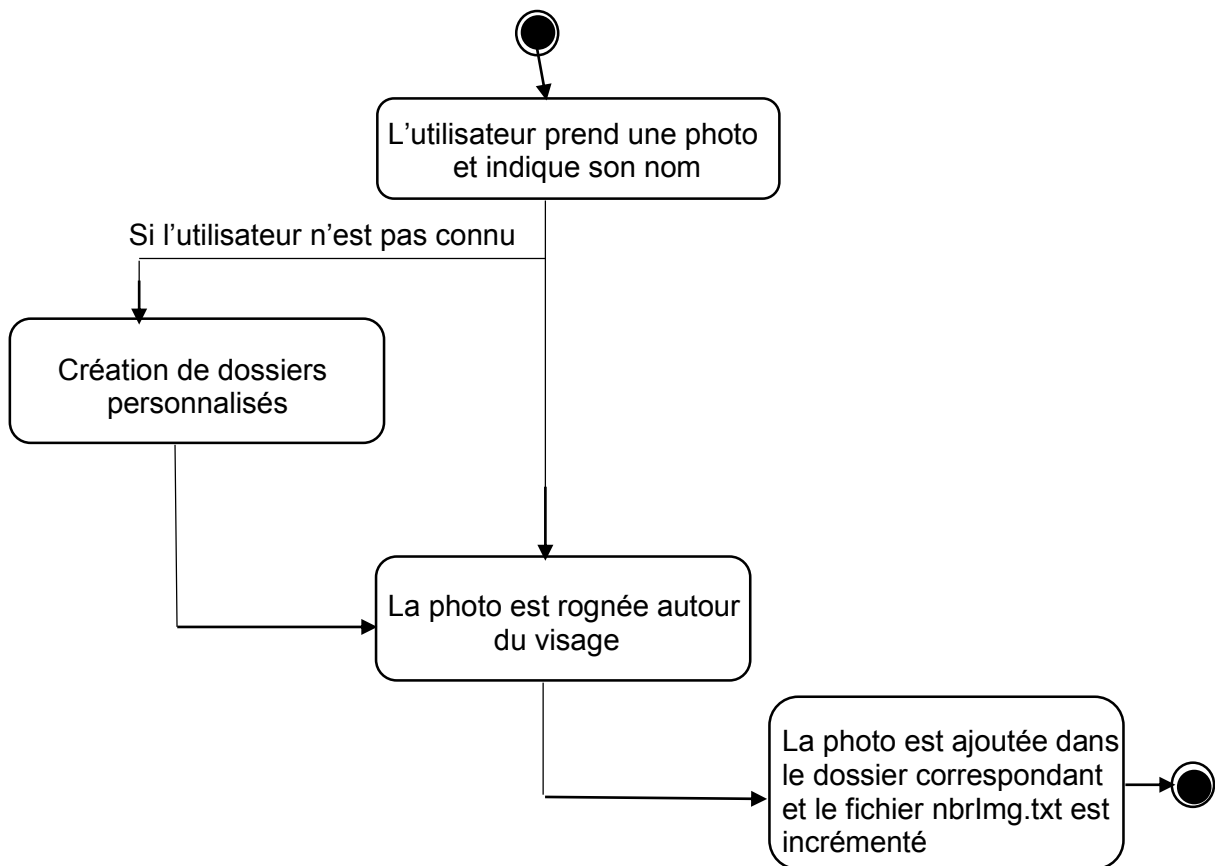
Après certaines recherches sur Internet pour comprendre comment fonctionne la reconnaissance faciale avec OpenCv, il s'est avéré que la version 2.3 d'OpenCv, installée en premier lieu pour la découverte de la bibliothèque, ne possédait pas les fonctions nécessaires pour la reconnaissance. Une solution a été de télécharger la version 2.4 d'OpenCv proposant les fonctions nécessaire. Or, les dépôts de Debian 7 ne proposent pas la version 2.4 de la bibliothèque. Pour obtenir les bons dépôts, il a fallu installer, sur une machine virtuelle (sur le PC sous GNU/Linux) avec Debian 8 (Virtual Box pour la machine virtuelle). L'installation de Debian 8 Jessie s'est faite avec les paramètres par défaut, et l'installation d'OpenCv se fait comme vu précédemment (Voir page 15).

L'apprentissage se fera à partir d'une commande vocale (prise de photos) et créera un ensemble de dossiers au nom de la personne. La reconnaissance se fera avec un parcours des images contenues dans ces dossiers. La bibliothèque OpenCv regroupe de nombreuses fonctions utilisables pour faire cela.

c. Mise en place de l'environnement du système

La bibliothèque OpenCv propose une fonction « FaceRecognition() » qui permet, avec un lot d'images, d'effectuer une correspondance entre ces images et un des visages détectés.

A chaque nouvelle personne est associé un dossier regroupant les images de la personne ainsi qu'un fichier avec le nombre d'image (utile pour la méthode de chargement des images car celle-ci charge exactement le nombre d'image donné).



```

root@debian:~/Documents/Josiane_Flo/Ressources# tree SaveImg/
SaveImg/
├── Florent
│   ├── Florent1.jpg
│   ├── Florent2.jpg
│   ├── Florent3.jpg
│   ├── Florent4.jpg
│   ├── Florent5.jpg
│   ├── Florent6.jpg
│   └── nbrImg.txt
└── Florent_Chauvette
    └── nbrImg.txt
2 directories, 8 files
  
```

Exemple d'agencement des dossiers des bases de données.

Un dossier au nom de la personne est créé si le nom rentré n'est pas connu. Ensuite, toute image associée au nom sera enregistrée dans le dossier correspondant. Puis le nombre d'images contenues dans le dossier est inscrit dans le fichier nbrImg.txt présent dans chaque dossier.

Les dossiers se trouveront dans /home/auto/Documents/Josiane/Ressources/SaveImg.

Pour l'apprentissage, il faut rentrer le nom de la nouvelle personne sur l'IHM (la reconnaissance vocale ne serait pas assez précise pour les noms propres et les prénoms) puis le programme vérifie que la personne n'existe pas dans la base de données pour éviter toute incohérence. La tête demande alors à la personne de prendre différentes poses, pour maximiser les taux de réussite, prends les photos et rogne les images de façon à ne garder que la tête et avoir la même taille pour toutes les images (la fonction d'OpenCv demande des images de mêmes tailles).

Pour la reconnaissance, les images sont chargées dans un vecteur de matrices dossiers par dossiers (sur l'exemple ci-dessus, toutes les images du dossier Florent seront chargées avant les images de Florent_Chauvette). Ensuite, deux étapes sont nécessaires : l'association d'images et labels, et la prédiction.

L'association d'images et labels consiste à associer un numéro (un identifiant) propre à la personne (Exemple : Florent aura le label « 1 », Florent_Chauvette aura le label « 2 », la prochaine personne aura le label « 3 » et ainsi de suite...). Pour associer un label à une image, il faut que l'image ait le même rang dans le tableau de vecteur de Mat que le label auquel elle correspond dans le vecteur d'entier. Un minimum de 5 images par personnes permet un taux de réussite de 60%.

	<i>Vecteur Images</i>	<i>Vecteur Labels</i>
<i>Rang 1</i>	Image P1	Label 1
<i>Rang 2</i>	Image P1	Label 1
<i>Rang 3</i>	Image P1	Label 1
<i>Rang 4</i>	Image P2	Label 2
<i>Rang 5</i>	Image P2	Label 2

Disposition des vecteurs pour l'association

La deuxième étape, la prédiction, permet de savoir, pour un visage détecté sur une image, de savoir à quelle personne, de la base de données, le visage s'apparente le plus.

Pour cela, une fonction d'OpenCv permet de renvoyer le label de l'image qui ressemble le plus au visage.

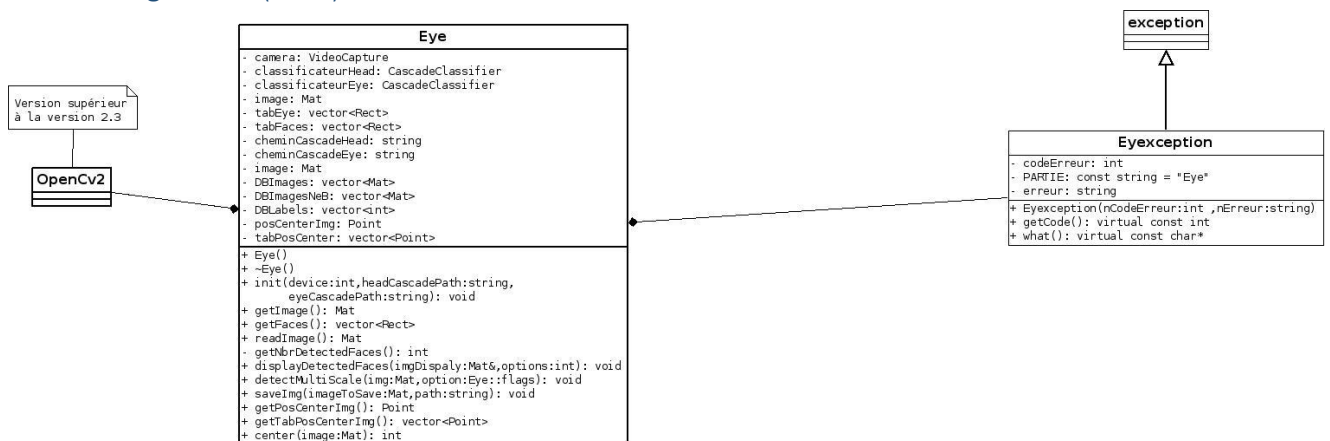
Le fichier « AssoLabels.txt » regroupe les différentes associations nom / labels existant. Ce fichier se trouve dans /home/auto/Documents/Josiane/Ressources/

d. Conclusion sur l'avancement du travail

Les prises de photos pour constituer la base de données personnelle se fait via un bouton de l'IHM. La reconnaissance est assez précise (~70% de réussite sur une quinzaine d'essais) malgré le fait qu'une personne non connue sera identifié comme la personne ressemblant le plus. (Ex : Bernard et Julie sont dans la base de données, Maxime n'y est pas, si Maxime ressemble plus à Bernard qu'à Julie il sera identifié comme Bernard). Toutes méthodes de gestions des bases de données sont opérationnelles sur un PC GNU/Linux sous Debian 8.

6. Classe Eye / Eyexception

a. Diagramme (UML)



(Le fichier .h et .cpp ainsi que les tests unitaires des 2 classes se trouvent dans les annexes)

La classe Eye est composée de la bibliothèque OpenCv2 et de la classe Eyexception qui dérive de la classe exception. Les erreurs de la classe Eye sont gérées par des exceptions de type Eyexception qui retourne un code d'erreur, ainsi qu'une chaîne de caractères indiquant la partie de la tête où se trouve l'erreur (en l'occurrence, l'œil) et un descriptif de l'erreur. La classe Eye n'est, à ce moment, pas complète, mais elle illustre l'avancement du projet.

b. Explications

Afin de pouvoir gérer un maximum d'erreurs, le constructeur Eye() ne fera rien d'autre qu'initialiser les variables de la classe. La fonction init(...) charge les fichiers CascadeClassifier et initialise le flux vidéo. Si une erreur est détectée, alors la fonction lève une exception de type Eyexception expliquant où se trouve l'erreur.

La fonction init(...) doit être appelée directement après le constructeur !

La fonction readImage() permet de capturer des images depuis la caméra. Enfermée à l'intérieur d'une boucle while(), on génère ainsi un flux vidéo.

La fonction displayDetectedFaces(Mat img, Eye::flags option) permet d'afficher les rectangles se trouvant autour des visages détectés. Le paramètre SeulVisage indique si le programme affiche le premier visage détecté seulement (true) ou s'il doit afficher tous les visages détectés (false). Le deuxième paramètre, precision, indique si le programme doit détecter les visages que s'il détecte des yeux à l'intérieur du visage car il se peut que, avec le fichier haarcascade_frontalface_alt2.xml, le programme détecte des visages n'existant pas. Afin de pallier ce problème, on peut faire appel au deuxième fichier .xml afin de ne détecter que les « vrais » visages. Le problème de cette solution est que le temps d'exécution est rallongé.

Le reste des méthodes sont développées dans la documentation Doxygen se trouvant dans les annexes.

c. Version Lite

La carte BeagleBone Black étant sous Debian 7, la version d'OpenCv se retrouve inférieure à 2.4 et n'inclue donc pas la reconnaissance faciale. De plus, les performances de la carte empêchent une utilisation optimale d'OpenCv. En effet, la détection de visages utilise un nombre important de ressources ce qui rend le traitement d'image assez lent.

C'est pourquoi une version « Lite » de la classe Eye, a été développée. Celle-ci regroupe les mêmes méthodes que la classe Eye complète mais les méthodes concernant la reconnaissance faciale (la reconnaissance elle-même, la gestion des bases de données personnelles...) sont retirées. La détection de visages précise (Retiens les visages SEULEMENT si des yeux sont détectés à l'intérieur) est aussi supprimée mais la détection de visage est toujours présente.

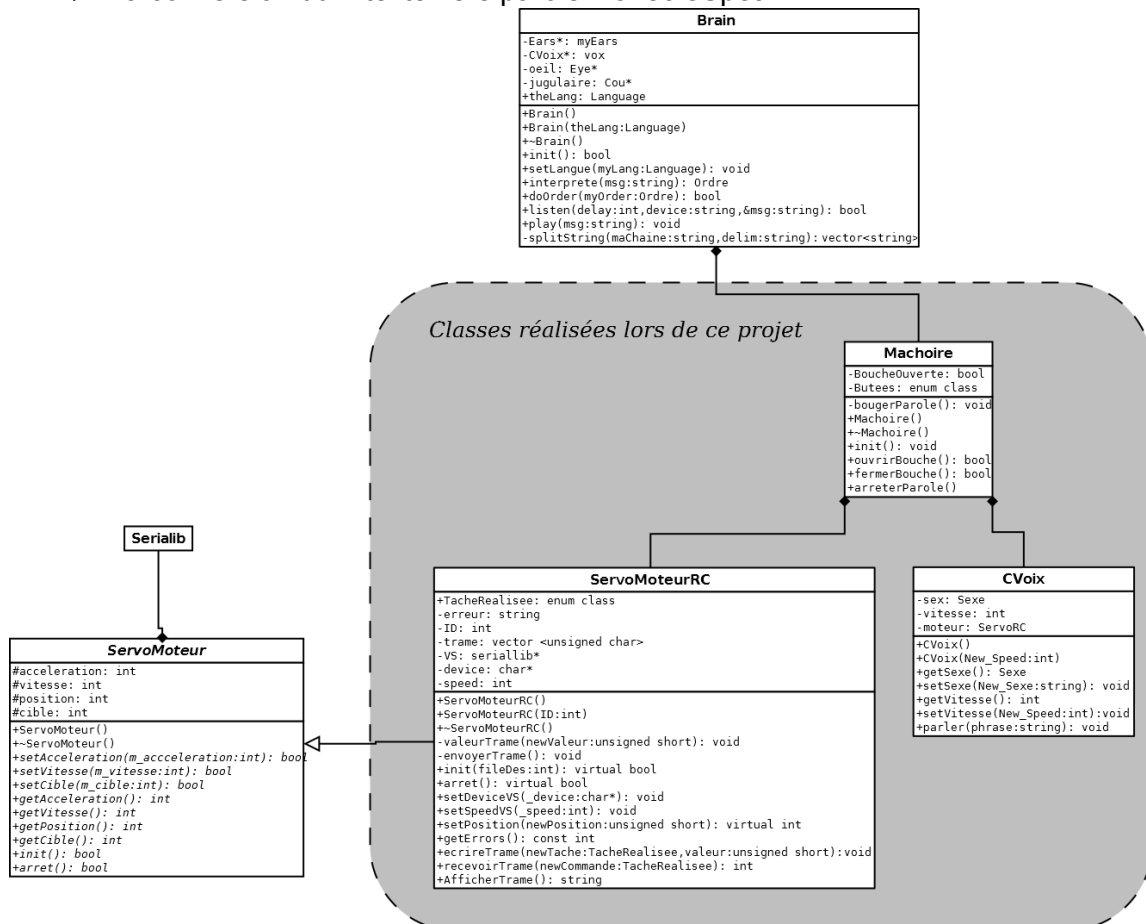
Eye
<ul style="list-style-type: none">- camera: VideoCapture- classificateurHead: CascadeClassifier- classificateurEye: CascadeClassifier- image: Mat- tabEye: vector<Rect>- tabFaces: vector<Rect>- cheminCascadeHead: string- cheminCascadeEye: string- posCenterImg: Point- tabPosCenterImg: vector<Point>
<ul style="list-style-type: none">+ Eye()+ ~Eye()+ init(device:int,headCascadePath:string,eyeCascadePath:string): void+ readImage(): Mat+ getFaces(): vector<Rect>+ getNbrVisagesDetect(): int+ displayDetectedFaces(imgDisplay:Mat&,options:int): void+ detectMultiScale(img:Mat,option:Eye::flags): void+ getImage(): Mat+ getCascade(option:Eye::flags): CascadeClassifier+ saveImg(imageToSave:Mat,path:string): void+ getPosCenterImg(): Point+ getTabPosCenter(): vector<Point>+ center(image:Mat): int- getCenter(): void

IV. LOUIS Nicolas : Commande des servomoteurs RC et Conversion « Text to Speech »

1. Rôle dans le projet

Dans ce projet, j'ai eu plusieurs tâches à réaliser :

- La commande des servomoteurs RC (mâchoire et yeux) à partir de la carte Pololu.
- La conversion du « texte vers parole » avec eSpeak.



2. Planification du projet

ACTIVITÉ	DÉBUT	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	FIN
Initialisation du projet																								
Test du matériel																								
Modélisation du projet																								
Documentation ServoMoteurs RC / Pololu																								
Documentation ESpeak / Mbrola																								
Trame de commande ServoMoteurs RC																								
Montage de la tête robotisée																								
Installation de Debian - Jessie																								
Installation Compilation Croisée																								
Conception de la classe CVoix																								
Problème carte son																								
Documentation Voie Série																								
Installation SerialLib																								
Conception de la classe ServoMoteurRC																								
Conception de la classe Machoire																								
Rédaction Rapport / Revue																								

3. La compilation croisée

a. Qu'est-ce que la compilation croisée ?

La compilation croisée, ou cross-compilation en anglais, est le fait de compiler des programmes pour une autre architecture, ou pour un autre système d'exploitation. Dans le cas de notre projet, cela est utilisé pour compiler un programme sur un ordinateur (sous Gnu/Linux – version Jessie) à destination de la BeagleBoneBlack (sous Gnu/Linux – architecture arm).

La raison principale de faire de la compilation croisée est la suivante :

- ✚ Avoir des puissances de calculs plus importantes (la compilation prendra moins de temps sur le PC que sur la BeagleBoneBlack).

b. Comment la mettre en place ?

- ✚ Installer un logiciel capable de compiler notre programme (ex: NetBeans)
- ✚ Savoir si notre système de destination supporte ou pas les architectures à virgule flottante : Si oui armhf, si non armel.
- ✚ Installer le logiciel curl (pour télécharger des données par syntaxe URL)
- ✚ Modifier le fichier des dépôts Debian
- ✚ Rajouter la signature des dépôts emdebian
- ✚ Taper les commandes pour installer le cross compilateur
- ✚ Intégration des chaînes de cross compilation dans NetBeans

Illustration :

```
root@debian:~$ apt-get install curl
root@debian:~$ nano /etc/apt/sources.list
root@debian:~$ curl http://emdebian.org/tools/debian/
emdebian-toolchain-archive.key | sudo apt-key add -
root@debian:~$ dpkg --add-architecture armhf
root@debian:~$ apt-get update
root@debian:~$ apt-get install crossbuild-essential-armel
```

c. Comment l'utiliser ?



On crée notre programme sous NetBeans en précisant dans les options l'architecture pour laquelle on va compiler (armhf). On lance la compilation afin de générer un exécutable. On envoie alors via SSH le programme sur la BeagleBoneBlack. On peut désormais exécuter celui-ci en ligne de commande.

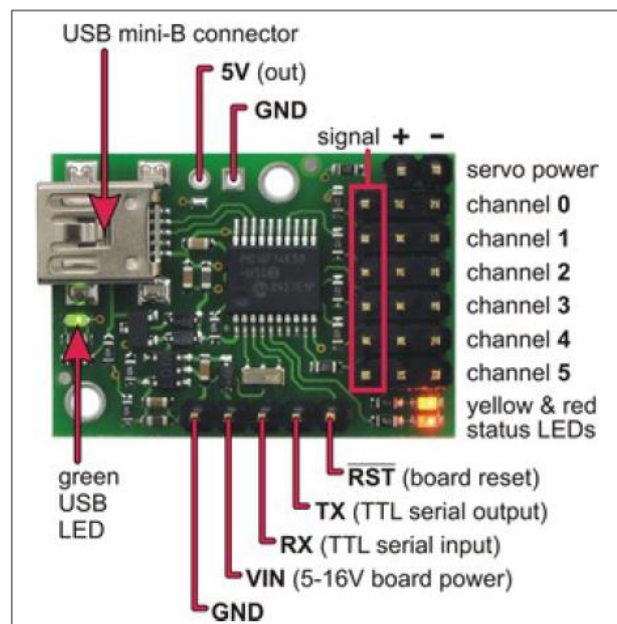
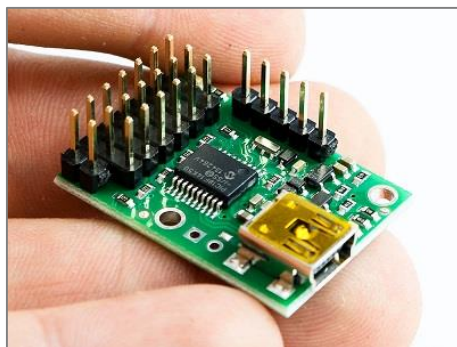
4. Carte MicroMaestro du constructeur Pololu

a. Qu'est-ce que la Pololu ?

La Micro Maestro est une carte électronique de Pololu de dimension 2.15mm * 30.5mm. Elle permet de commander des servomoteurs de type Radio Commande (RC) et d'effectuer une communication en TTL 5V (Transistor-Transistor Logic). Un connecteur USB permet de la relier à un contrôleur tel qu'un PC avec un débit de 200Kb/s maximum. C'est par ce port que la carte est également alimentée.

Sur cette carte, il y a 6 canaux générateurs d'impulsions pour les servomoteurs. Chacun d'entre eux peut être contrôlé individuellement et configurable sur plusieurs critères afin de faire varier la position, la vitesse et l'accélération angulaire de l'axe du servomoteur. Les servomoteurs ne sont pas alimentés par la carte mais par une batterie externe de 7,4V.

Il y a également présence sur cette carte de 5 pins permettant la réalisation de la voie série, mais dans notre cas seulement trois nous sont utiles (RX, TX, Ground).

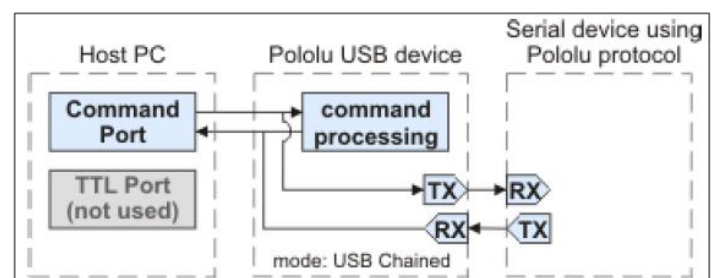


b. Quels sont les modes de connexion ?

Trois modes de connexion peuvent être utilisés entre la carte Micro Maestro et le PC :

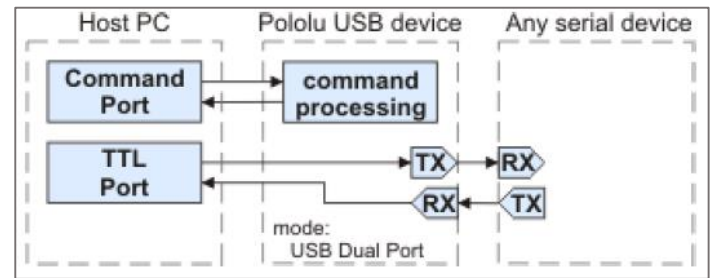
▪ Le mode Chainé :

Ce mode de connexion est utilisé pour envoyer simultanément des commandes à la carte Pololu (USB) et des commandes sur la Voie Série via la ligne TX. Il permet d'avoir qu'un seul port COM et de brancher plusieurs Pololu en série (qui utiliseront un protocole compatible). Le TTL n'est pas utilisé dans le cas présent.



- Le mode Dual-Port :

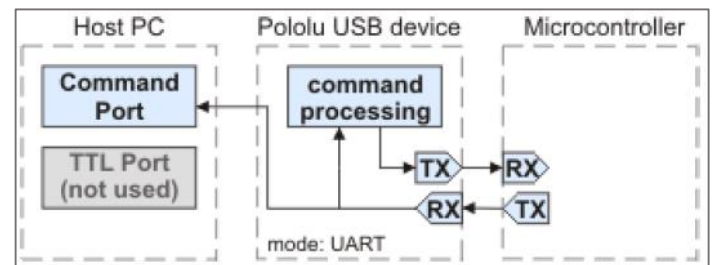
Comme son nom l'indique, ce mode utilise deux ports. Un pour envoyer la commande, et l'autre pour la gestion du TTL. La trame est envoyée du PC vers la Pololu, puis une réponse lui est alors retournée. Le port TTL peut être utilisé mais cela est inutile dans notre cas.



- Le mode UART :

(Universal Asynchronous Receiver Transmitter = émetteur-récepteur asynchrone universel) :

La pololu est ici contrôlée par voie Série à partir d'un microcontrôleur. Ce mode n'est donc pas utile à la réalisation de notre projet.



Pour ma part, j'ai choisi d'utiliser le mode « Dual-Port », car vu que l'on utilise seulement le mode commande, cela simplifie notre schéma à ceci :

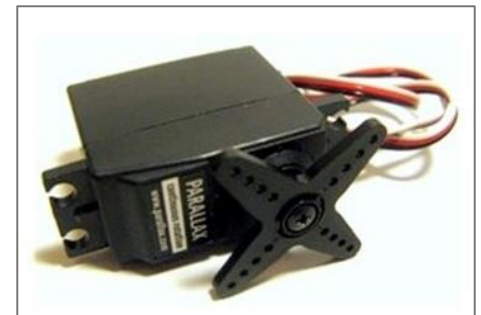
c. Les Servomoteurs de type RC

- Définition :**

Les servomoteurs de type RC (Radio Control) sont des moteurs « non-intelligents » comparés aux moteurs « HerkuleX » ou « Dynamixel ». Ils fonctionnent en MLI (Modulation à Largeur d'Impulsion). Ils ne sont pas capables de se positionner eux même. Afin de pouvoir donner ou connaître la position d'un servomoteur RC, il faut regarder la largeur d'impulsion de ce dernier. Chaque largeur d'impulsion correspond à une position du servomoteur.

Comme par exemples :

- 1 ms → target = 1000 → position buté gauche
- 1, 5 ms → target = 1500 → position centrale
- 2 ms → target = 2000 → position buté droite



- La trame :**

Afin de contrôler les servomoteurs, il faut envoyer une trame à la Pololu. Cette trame fixera le numéro du servomoteur à commander et le paramètre que l'on veut modifier (la position, la vitesse de déplacement, l'accélération).

Cette trame est composée de 4 octets. Afin d'expliquer clairement comment est constituée celle-ci, je vais le faire sous forme d'un exemple. Voyons celui-ci :

Trame : 0x84 0x00 0x70 0x2E

- Avec : - 0x84 → Valeur de la commande à utiliser (ici setTarget)
- 0x00 → Numéro du Servomoteur à contrôler
- 0x70 0x2E → Valeur du target en quart de µs

Calcul du Target :

Cas de la position centrale (Target = 1500)

On multiplie 1500 par 4 afin de l'avoir en µs → $1500 * 4 = 6000$

On passe le résultat en binaire → 00101110 01110000 (Le MSB de chaque octet est à 0)

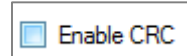
Conversion en Hexadécimal → 0x2E 0x70

Inversion des deux octets → 0x70 0x2E

▪ Le CRC-7 :

Le CRC (Cyclic Redundancy Check = Contrôle de Redondance Cyclique) est un outil logiciel permettant de détecter les erreurs de transmission ou de transfert des données. Celui-ci est optionnel. Dans le logiciel constructeur, une case peut-être cochée afin de définir si l'on veut ou pas le CRC dans notre frame.

Il se calcule de la façon suivante :



- ✚ Exprimer les 8-bit du polynôme en binaire en commençant par le bit de poids faible. Le polynôme 0x91 (0x91 est le polynôme adapté à notre CRC) est écrit comme ceci : 10001001.
- ✚ On ajoute alors 7 zéros à la fin de notre message.
- ✚ On écrit notre polynôme sous le message à envoyer, ainsi, le LSB de notre polynôme est directement sous le LSB de notre message.
- ✚ Si le LSB du polynôme est aligné sous un 1, on fait un XOR (OU Exclusif) entre le polynôme et le message pour obtenir un nouveau message
- ✚ Si le LSB de notre polynôme est aligné sous un 0, on ne fait rien.
- ✚ On décale alors notre polynôme d'un bit à droite.
- ✚ Si tous les 8 bits du polynôme sont encore alignés avec des morceaux du message, on retourne à l'étape où l'on positionne le polynôme sous le LSB.
- ✚ Ce qui reste du message est maintenant le résultat du CRC-7. On transmet ces 7 bits comme octet de CRC.

Pour le moment, le CRC n'est pas intégré dans mon projet. J'ai juste recherché les informations le concernant (Définition, méthode de calcul) et j'ai ensuite réalisé son algorithme. Il sera intégré si le temps me le permet puisqu'il est facultatif.

Illustration :

```

Steps 1 & 2 (write as binary, least significant bit first,
add 7 zeros to the end of the message):

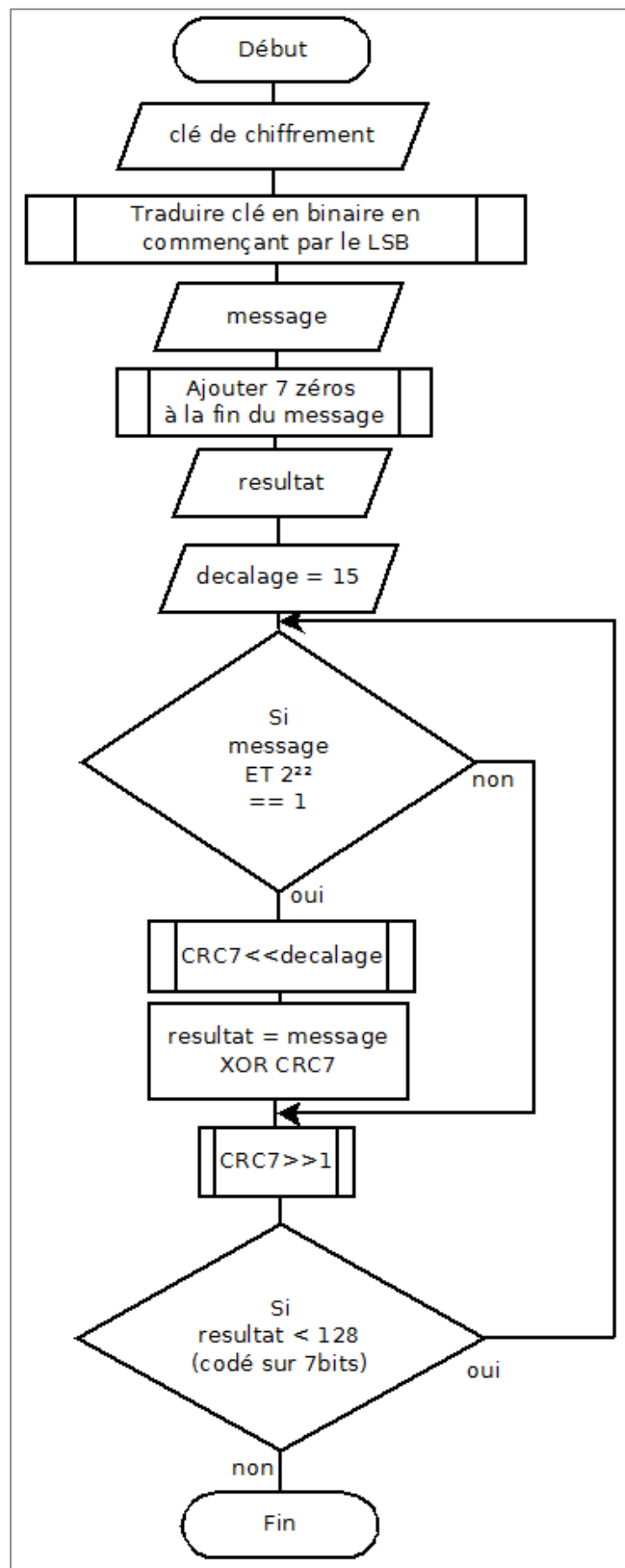
CRC-7 Polynomial = [1 0 0 0 1 0 0 1]
message = [1 1 0 0 0 0 0 1] [1 0 0 0 0 0 0 0] 0 0 0 0 0 0 0

Steps 3, 4, & 5:

1 0 0 0 1 0 0 1 ) 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                XOR 1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                ----- | | | | | | | | | | | | | | | | | |
                  1 0 0 1 0 0 0 1 | | | | | | | | | | | | | | |
shift  -----> 1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                  ----- | | | | | | | | | | | | | | | | | |
                    1 1 0 0 0 0 0 0 | | | | | | | | | | | | | | |
                    1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                    ----- | | | | | | | | | | | | | | | | | |
                      1 0 0 1 0 0 1 0 | | | | | | | | | | | | | | |
                      1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                      ----- | | | | | | | | | | | | | | | | | |
                        1 1 0 1 1 0 0 0 | | | | | | | | | | | | | | |
                        1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                        ----- | | | | | | | | | | | | | | | | | |
                          1 0 1 0 0 0 1 0 | | | | | | | | | | | | | | |
                          1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                          ----- | | | | | | | | | | | | | | | | | |
                            1 0 1 0 1 1 0 0 | | | | | | | | | | | | | | |
                            1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                            ----- | | | | | | | | | | | | | | | | | |
                              1 0 0 1 0 1 0 0 | | | | | | | | | | | | | | |
                              1 0 0 0 1 0 0 1 | | | | | | | | | | | | | | |
                              ----- | | | | | | | | | | | | | | | | | |
                                1 1 1 0 1 0 0 0 = 0x17

```

Algorithme illustrant le calcul du CRC vu à la page précédente :



d. La classe ServoMoteurRC

Afin de contrôler un servomoteur RC, j'ai réalisé la classe ServoMoteurRC. Elle hérite de la classe abstraite « Servomoteur ». Cela signifie donc qu'elle regroupe les méthodes communes entre ServoMoteurRC et la classe ServoDyn.

Cette classe ServoMoteurRC va nous permettre de composer la trame à envoyer à la Pololu, d'envoyer cette même trame, d'initialiser un moteur, d'arrêter un moteur, de changer les valeurs de position, de vitesse et d'accélération mais également de récupérer ces valeurs. Chaque méthode est bien détaillée en annexe grâce à la documentation Doxygen réalisée.

Mais afin d'échanger entre la pololu et la BeagleBone, une voix série a été mise en place. Pour cela, comme Kevin Stoeber, nous utilisons la bibliothèque Seriallib qui va nous permettre d'effectuer ces connexions. Pour l'utiliser, il suffit simplement de la télécharger et de l'inclure dans le projet à compiler.

Seriallib est téléchargeable à l'adresse suivante : <http://seriallib.free.fr/html/index.html> Une fois le dossier téléchargé, il faut en extraire les fichiers .cpp et .h. Ces deux fichiers doivent être copier/coller dans le projet ServoMoteurRC. On reouvre ensuite ServoMoteurRC.h sous NetBeans et nous pouvons rajouter.

La classe Seriallib offre un accès simple à des dispositifs de port série pour Windows et Linux. Il peut être utilisé par tout périphérique série (Built-in port série, convertisseur USB vers RS232, carte Arduino ou de tout matériel utilisant ou émulant un port série).

«Ceci est un logiciel sans licence, il peut être utilisé par toute personne qui essaie de construire un monde meilleur.»

a. Qu'est-ce que le Text-To-Speech ?

Le Text to speech (TTS), est une technique informatique permettant de transformer un texte écrit en parole artificielle, autrement dit, c'est de la synthèse vocale. Il existe plusieurs solutions libres de synthèse vocale : Festival et eSpeak sont les deux plus connues fonctionnant sous linux... Dans le but de mon projet, je dois utiliser eSpeak. Celui-ci permet de réaliser une lecture « grossière » du texte, c'est pourquoi j'ai décidé de le coupler avec le logiciel Mbrola, permettant ainsi de lire le texte avec des phonèmes. La lecture est moins hachée et donc plus fluide. Le résultat est beaucoup plus appréciable.

e. La classe Machoire

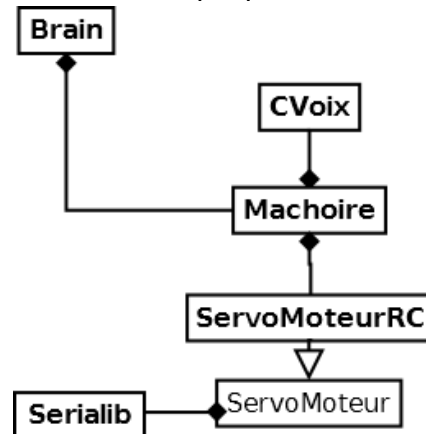
La classe Machoire est créée afin de faire bouger la mâchoire. Elle est donc composée de ServoMoteurRC mais également de notre autre classe CVoix, car Machoire regroupe les méthodes suivantes : ouvrirBouche, fermerBouche, bougerParole et arreterParole. Ouvrir et fermer bouche vont permettre toutes les deux la simple ouverture ou fermeture de la bouche. Il s'agit donc de faire bouger le servomoteur jusqu'à la position en butée voulue. Ces valeur ont été calculé préalablement et sont définies constantes dans notre code afin qu'elles ne puissent pas être modifiées. BougerParole va permettre, grâce à un thread, de faire bouger la mâchoire pendant l'intégralité du temps de parole. Il s'agit simplement de lui faire ouvrir et fermer la boucher alternativement. Une fois le thread terminé, la méthode fermerBouche et de nouveau appelée afin d'être sûr de la position du servomoteur. ArreterParole va nous permettre d'interrompre Josiane si besoin est. Pour cela je lui fixe une nouvelle chaine à dire. Cette chaîne étant vide, elle va donc s'arrêter. Tout comme la fonction bougerParole, je termine par un fermerBouche afin de retourner en position initiale.

ServoMoteurRC
+TacheRealisee: enum class -erreur: string -ID: int -trame: vector <unsigned char> -VS: seriallib* -device: char* -speed: int +ServoMoteurRC() +ServoMoteurRC(ID:int) +~ServoMoteurRC() -valeurTrame(newValeur:unsigned short): void -envoyerTrame(): void +init(fileDes:int): virtual bool +arret(): virtual bool +setDeviceVS(_device:char*): void +setSpeedVS(_speed:int): void +setPosition(newPosition:unsigned short): virtual int +getErrors(): const int +ecrireTrame(newTache:TacheRealisee,valeur:unsigned short): void +recevoirTrame(newCommande:TacheRealisee): int +AfficherTrame(): string

Machoire
-BoucheOuvrte: bool -Butees: enum class -bougerParole(): void +Machoire() +~Machoire() +init(): void +ouvrirBouche(): bool +fermerBouche(): bool +arreterParole()

Afin de bien visualiser l'utilité de mes classes, voici un diagramme de classe de ma partie.

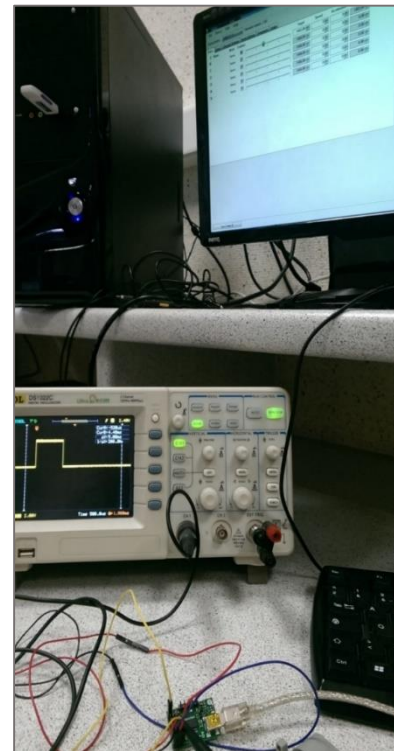
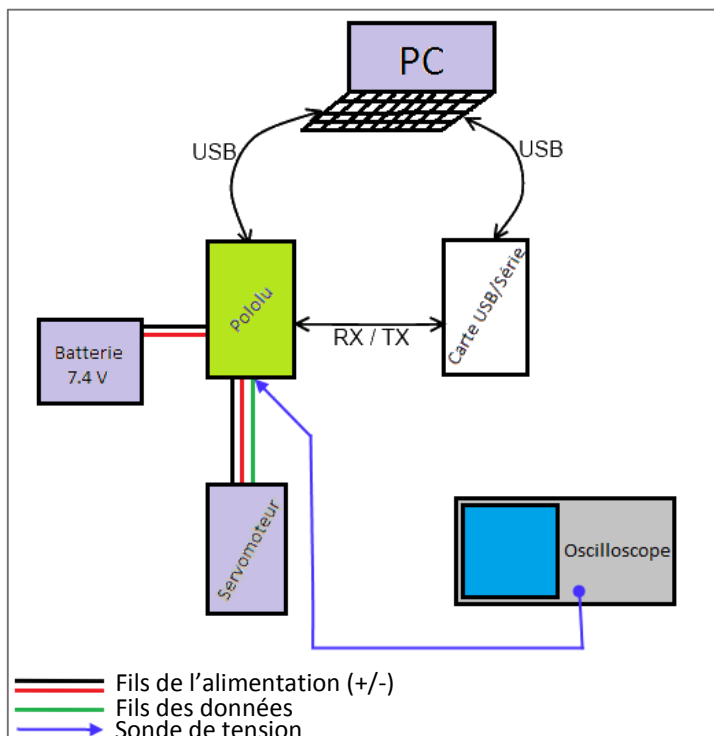
- ✚ Cela nous permet de voir que ServoMoteur est composé de Serialib. Insertion réalisée à ce niveau-là, vu que celle-ci est utilisée par ServoDyn et ServoMoteurRC.
- ✚ Nous pouvons également constater qu'il s'agit d'un héritage entre ServoMoteur et ServoMoteurRC.
- ✚ La classe Machoire est quant à elle composée de CVoix et de ServoMoteurRC. CVoix réalise principalement la fonction parler() de notre tête et ServoMoteurRC permet le contrôle des ServoMoteur fonctionnant en modulation par largeur d'impulsions. La classe Machoire va donc être l'union de ces deux classes, qui à l'aide d'un thread, va permettre de faire bouger la mâchoire en même temps que le son sort du Haut-parleur.

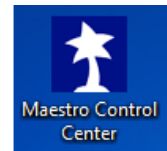
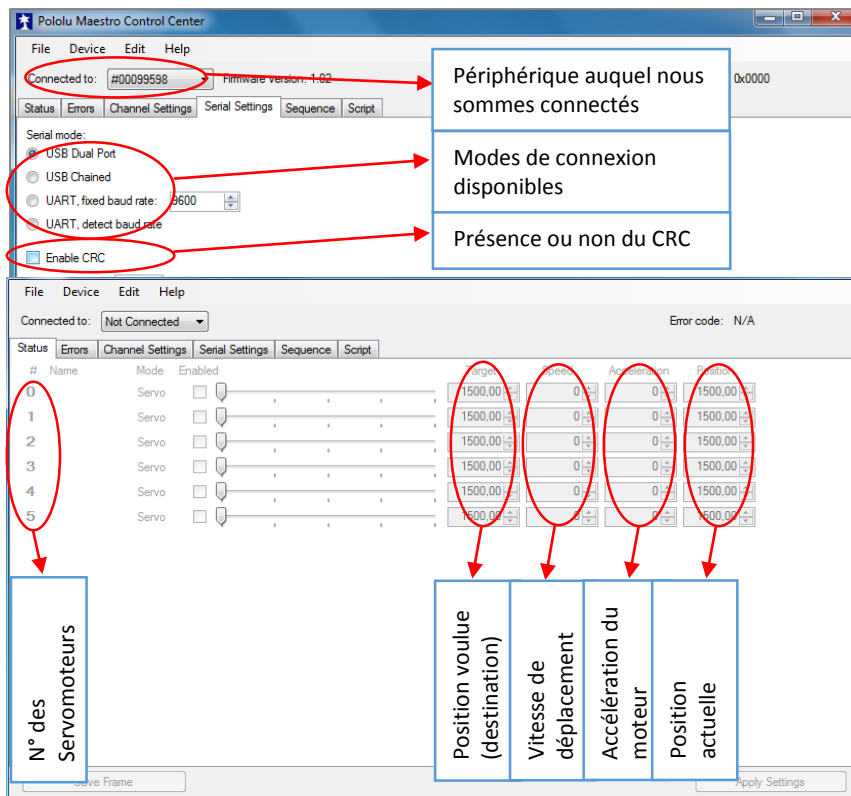


f. Tests unitaires

Au début de notre projet, il a fallu comprendre comment fonctionne réellement les servomoteurs de type RC, car ceux-ci vont être utilisés dans le contrôle des yeux et de la mâchoire. Nous avons donc réalisé un montage permettant de visualiser concrètement les largeurs d'impulsions.

Nous pilotons les servomoteurs branchés sur la Pololu grâce au logiciel constructeur Maestro.

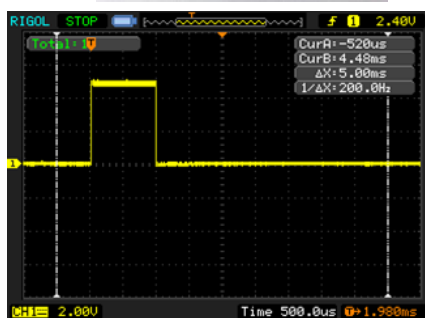




Ce logiciel, Maestro Control Center, est fourni par la société Pololu. Il s'agit d'une Interface Homme Machine (IHM) nous permettant le contrôle des Servomoteur connectés à la carte Pololu.

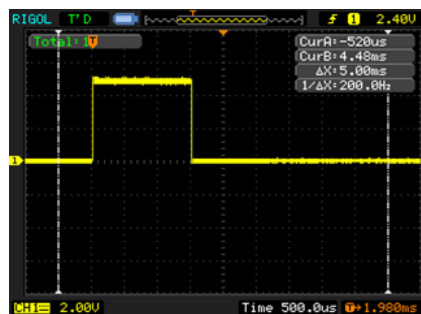
Nous avons obtenu les résultats suivants :

Target = 1000
Position gauche



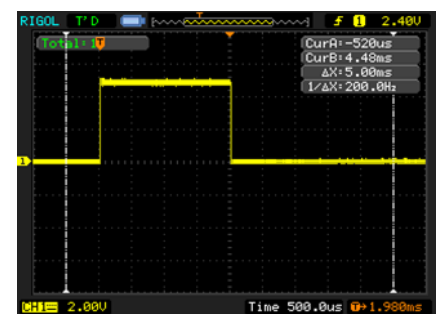
Période = 20 ms
Largeur d'Impulsion : 1 ms

Target = 1500
Position centrale



Période = 20 ms
Largeur d'Impulsion : 1,5 ms

Target = 2000
Position droite



Période = 20 ms
Largeur d'Impulsion : 2 ms

Codage émission des trames de commande des servomoteurs :

```
--- DEBUT DU PROG ---
Initialisation...
--> OK !
SetPosition
valeur souhaitee ? :
1000
Fermeture en cours...
--> OK !
--- FIN DU PROG ---

RUN FINISHED; exit value 0; real time: 22s; user: 0ms; system: 0ms
```

```
--- DEBUT DU PROG ---
Initialisation...
--> ERROR !

RUN FINISHED; exit value 1; real time: 10ms; user: 0ms; system: 0ms
```

5. ESpeak

a. Contrainte de la BeagleBone Black

La BeagleBone Black ne possède pas de sortie audio standard (prise Jack 3.5). Par défaut la sortie du son est réalisée par le mini HDMI, mais cela ne nous convenait pas. Nous avons alors décidé de prendre un adaptateur USB vers jack femelle. Cet adaptateur possède alors une prise micro ainsi qu'une prise Haut-parleur. Seule la sortie Haut-parleur est utilisée car notre micro est en USB.

Comme je l'ai marqué préalablement, la sortie son par défaut est la prise mini HDMI. Il a donc fallut rediriger celui-ci. J'ai alors crée un fichier permettant qu'au démarrage de la BeagleBone la carte son par défaut soit l'adaptateur USB. J'ai alors cherché le numéro de la carte son à utiliser ; pour cela j'ai ouvert sur nano le fichier `/proc/asound/cards`.

Le fichier `cards` ressemble à cela :

```
0 [HDMI      ]: HDA-Intel - HDA ATI HDMI
HDA ATI HDMI at 0xfe9ec000 irq 87
1 [Ultra     ]: USB-Audio - Fast Track Ultra
M-Audio Fast Track Ultra at usb-0000:00:16.2-3, high speed
```

La carte Son à prendre par défaut est donc la carte numéro 1.

Je peux désormais créer mon fichier de paramétrage par défaut. Au démarrage de la BeagleBone Black, Alsa vérifie l'existence d'un fichier de configuration :

- soit dans le répertoire qui s'applique alors à tous les utilisateurs (`/etc`). Son nom doit être : **asound.conf**

- soit dans le répertoire de l'utilisateur (`/usr`). Son nom doit être **.asoundrc**

Normalement, et par défaut, ces fichiers n'existent pas. Il faut donc utiliser l'éditeur de texte nano pour le créer et lui donner le bon nom (sans extension).

Dans tous les cas, le contenu de ce fichier doit être le suivant :

```
defaults.ctl.card 1
defaults.pcm.card 1
defaults.timer.card 1
```



b. La classe CVoix

Mr VEITES a réalisé dans sa partie une classe Brain qui est composée de l'ensemble des classes utiles sur la tête de l'humanoïde. Il s'agit de la classe « mère » de notre projet, elle regroupera l'ensemble des autres classes. Elle permet donc de contrôler l'ensemble des parties de la tête (Servomoteurs du coup (Dynamixel), des yeux et de la mâchoire (RC), les caméras, micro, et haut-parleur). Afin que l'accès au haut-parleur soit possible j'ai donc réalisé une classe Voix permettant de faire parler notre robot. Cette fonction est réalisée par un appel system. Nous composons la chaine de la façon suivante :

```
Chaine = « Espeak-s -v mb/mb-fr4 ChaineALire »  
System(chaine.c_str()) ;
```

Après avoir travaillé en binôme avec lui, nous avons pu déterminer les attributs et méthodes qui lui seraient utiles :

CVoix
-sex: Sexe -vitesse: int
+CVoix() +CVoix(New_Speed:int) +getSexe(): Sexe +setSexe(New_Sexe:string): void +getVitesse(): int +setVitesse(New_Speed:int): void +parler(phrase:string): void

c. Tests unitaires

Lorsque j'ai commencé à travailler sur la bibliothèque eSpeak, j'ai trouvé un logiciel simple utilisant celle-ci. Il s'agit d'un logiciel Open Source, téléchargeable à l'adresse : <http://espeak.sourceforge.net/>, qui m'a permis de comprendre l'intérêt d'eSpeak, mais surtout de visualiser l'ensemble des attributs et méthodes que je pourrais mettre dans ma classe.

Voici une capture d'écran de ce logiciel :

Position de la bouche : Utilisable dans une version future.

Voix : Nous n'avons inséré que la langue française pour le moment mais le choix entre une voix masculine et féminine est possible.

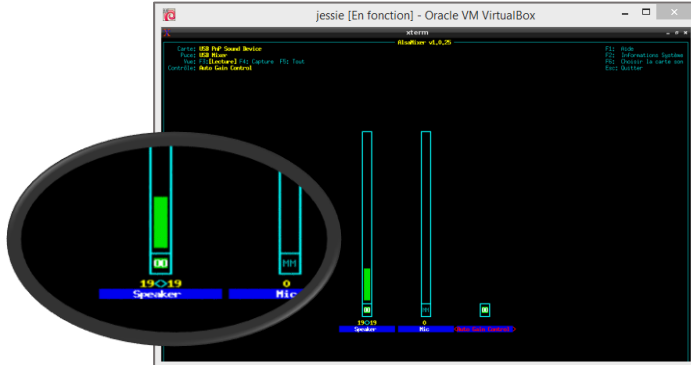
Vitesse : Possibilité de régler la vitesse de parole.

Une chaine : Texte que le Robot va dire

Parler : Fonction principale d'eSpeak permettant de faire parler le robot.

Dans notre projet, les options non entourées ne seront pas utilisées (Lecture de fichier texte ou audio, Start/Stop, Sauvegarder en fichier .wav ou encore Reset). Le volume n'a pas été pris en compte car il est géré directement sur la BeagleBoneBlack avec la commande alsamixer.

Alsamixer :



Test de la classe (fonction « Parler ») :

```
xterm
root@beaglebone:~/Bureau/test# ./cvoix-new 2> /dev/null
--- DEBUT PROGS ---
Creation de CVoix
Fct Parler
--- FIN PROGS ---
root@beaglebone:~/Bureau/test#
```

6. Conclusion

- ✓ La classe CVoix est entièrement fonctionnelle même si elle est améliorable avec inclusion de la classe eSpeak (suppression des appels system)
- ✓ La classe ServoMoteurRC est fonctionnelle
- ✓ La classe Machoire est fonctionnelle
- ✗ La classe Eye n'intègre pas encore les servomoteurs
- ✗ Problème rencontré avec les yeux. Ils ont beaucoup de butées à prendre en compte et il faut contrôler deux ServoMoteurRC en parallèle pour le pilotage d'un seul œil.

V. STOEBNER Kévin : Commande des servomoteurs Dynamixel

1. Rôle dans le projet

Gestion des servomoteurs du cou en fonction de la vision et de l'écoute : Mouvement simple du cou ou mouvement spécifique demandé.

2. Servomoteur Dynamixel : MX28-T

a. Présentation du servomoteur

Les servomoteurs du cou, dont je m'occupe dans projet sont des servomoteurs de la marque Dynamixel, le modèle MX28T. Les informations relatives à ces servomoteur ce trouve sur le site support du constructeur, Robotis :

http://support.robotis.com/en/product/dynamixel/mx_series/mx-28.htm

Voici quelques-unes de ses caractéristiques données sur ce site :

- Protocol Type
 - MX-28T (Half duplex Asynchronous Serial Communication (8bit, 1stop, No Parity))
- Link (Physical)
 - MX-28T (TTL Level Multi Drop Bus)
- ID : 254 ID (0~253)



Comme on peut le voir, ils utilisent une liaison Transistor Transistor Logic(TTL) pour communiquer. Avec un protocole half-duplex, asynchrone avec 1 bit de start, 8 bits de données, un bit de stop et pas de parité.

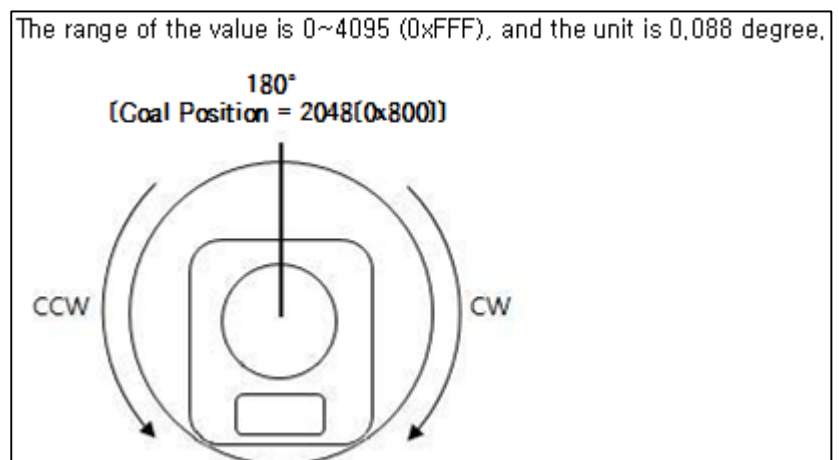
Les servomoteurs possède un identifiant compris entre 0 et 253.

- Running Degree
 - 0° ~ 360°
 - Endless Turn

Ils sont utilisables en deux modes, rotation « illimité » et rotation « 0-360° », ici la rotation « 0-360° » sera utilisée.

Le mode « 0-360° » correspond à des valeurs entières qui sont 0-4095. Comme on peut le voir sur l'image ci-contre cela fait un coefficient de proportionnalité de 0.088 entre les degrés et les valeurs entières.

- No load speed
 - 50rpm (at 11,1V)
 - 55rpm (at 12V)
 - 67rpm (at 14,8V)



Différent voltage sont possible, le 12V sera choisi dans le projet, limitant alors la rotation à 55 rpm (rotation par minute).

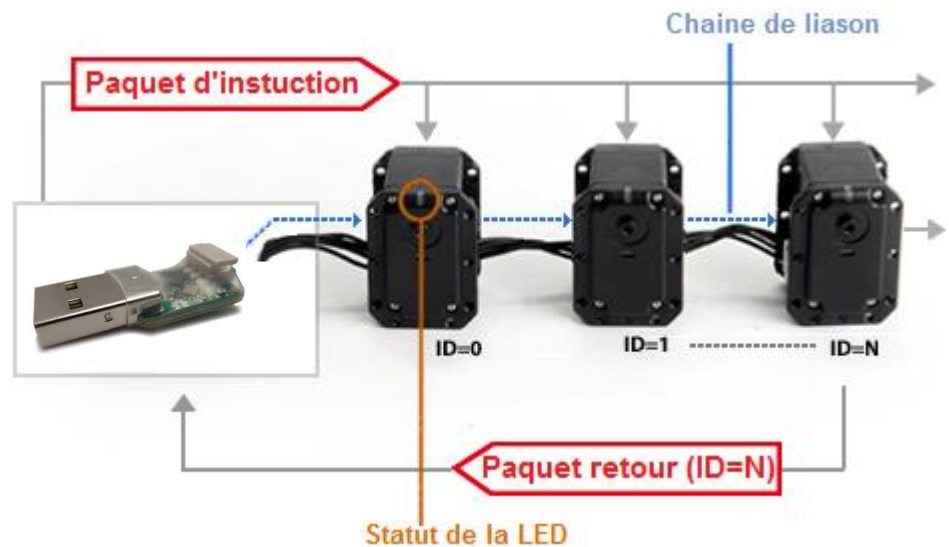
b. Communication avec les servomoteurs

Les servomoteurs utilisent, comme vu précédemment, une liaison TTL. Dans ce projet je communiquerais avec lui à une vitesse de 115 200 bauds.

Les servomoteurs utilisent un protocole « Maître-esclaves ».

C'est-à-dire qu'ils ne peuvent pas émettre tant qu'il n'y a pas eu de demande venant du programme.

Ils sont reliés en chaine comme le montre ce schéma. Les identifiants pouvant être placé dans n'importe quelle ordre cela n'a pas d'importance puisque le « maître » transmet à tous les esclaves en même temps.



La communication avec les servomoteurs se fait grâce à des trames. Ces trames utilisent au minimum 5 octets représentés en valeurs hexadécimale dont voici le format d'envoi et de réception :

Trame d'envoi

En-tête		N° Servo	Taille	Instruction	Donnée 1	...	Donnée n	Checksum
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 6	Byte x	Byte n	Byte n+1

Trame de réception

En-tête		N° Servo	Taille	Erreur	Donnée 1	...	Donnée n	Checksum
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 6	Byte x	Byte n	Byte n+1

Ces trames commencent toutes par un en-tête de : 0xFF 0xFF

- ✚ **N° Servo** correspond à l'identifiant du servomoteur sauvegardé dans la mémoire de celui-ci.
- ✚ **Taille** comptabilise le nombre de données + 2.
- ✚ **Instruction** permet de sélectionner l'écriture, la lecture de données.
- ✚ **Erreur** renvoi une erreur survenue lors de l'envoi de la trame ou une erreur sur le servomoteur.
- ✚ **Le tableau de données** contient les informations pour le servomoteur ou provenant de celui-ci.
- ✚ **Checksum** permet de détecter une erreur dans la trame. Il se calcul en ajoutant les octets de données et en inversant le tout.

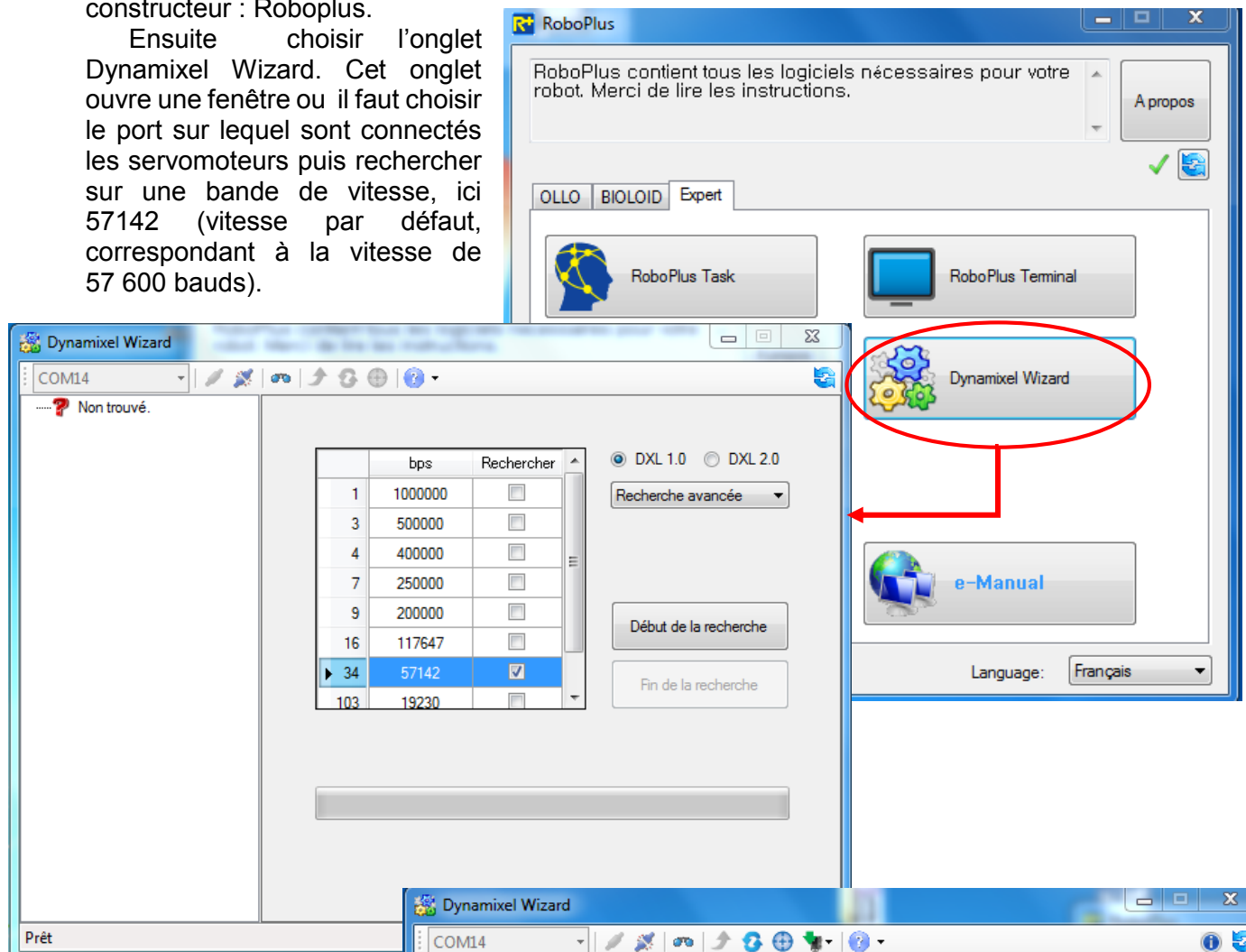
Ces informations ont été trouvées grâce à l'entreprise qui a créé un site d'explications du protocole à utiliser : http://wiki.cybedroid.com/index.php?title=Protocole_dynamixel_Robotis

3. Configuration

a. Configuration des servomoteurs

Pour configurer les servomoteurs, il a fallu passer par le logiciel proposé par le constructeur : RoboPlus.

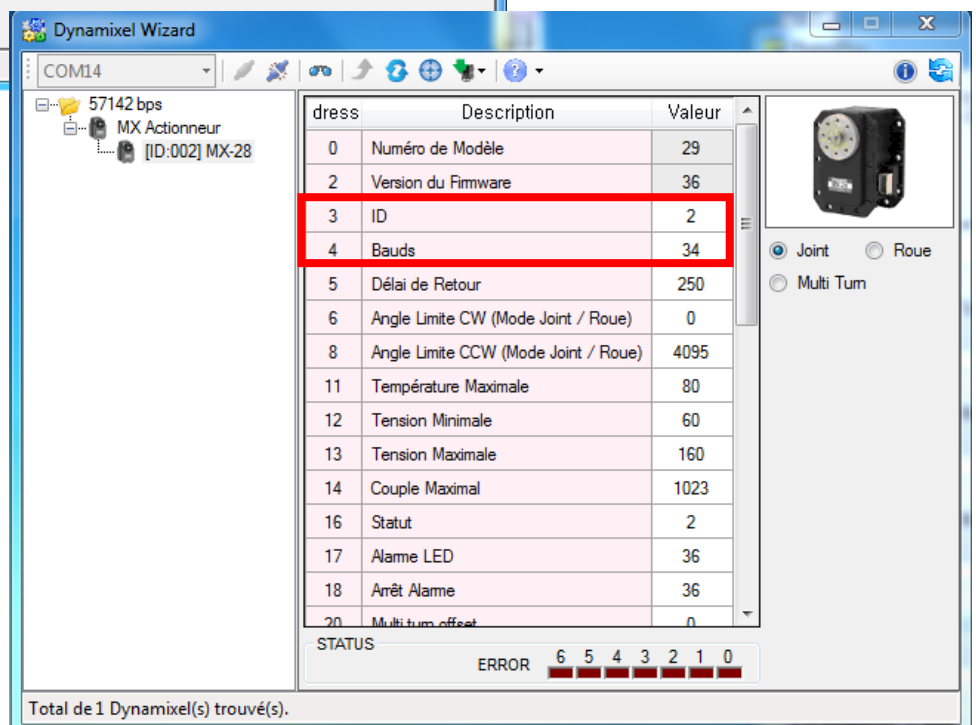
Ensuite choisir l'onglet Dynamixel Wizard. Cet onglet ouvre une fenêtre où il faut choisir le port sur lequel sont connectés les servomoteurs puis rechercher sur une bande de vitesse, ici 57142 (vitesse par défaut, correspondant à la vitesse de 57 600 bauds).



Une fois la recherche lancée, les servomoteurs sont détectés. Les servomoteurs avaient tous le même identifiant par défaut : 1. L'identifiant et la vitesse de chacun était donc à changer.

Pour la vitesse le logiciel proposait un menu déroulant avec des numéros correspondants à une vitesse.

Sur l'image ci-contre j'ai fixé l'ID à 2 et la vitesse à 115 200 avec le numéro 34 du menu déroulant.



b. Configuration de l'USB2AX

Pour communiquer avec les servomoteurs, j'utilise l'USB2AX.

L'USB2AX est un connecteur USB pour piloter les servomoteurs Dynamixel AX ou MX communiquant en TTL.

Or celui-ci est configuré pour fonctionner sous Windows. Étant donné que la réalisation de la classe tout comme le fonctionnement final sur la BeagleBone Black fonctionne sous Linux, il a fallu le configurer.



Pour ce faire, j'ai installé dfu-programmer sur linux. Dfu-programmeur est un programmeur multi-plate-forme en ligne de commande pour chips Atmel (8051, AVR, XMEGA & AVR32) avec un bus USB bootloader soutenant ISP. Les informations ci-dessus se trouvent sur ce site : <https://dfu-programmer.github.io/>

La configuration de l'USB2AX avec dfu-pogrammer a permis d'installer le bon chip pour l'utiliser sous linux.

```
root@iris-035:~# aptitude install dfu-programmer
Les NOUVEAUX paquets suivants vont être installés :
  dfu-programmer
0 paquets mis à jour, 1 nouvellement installés, 0 à enlever et 187 non mis à jour.
Il est nécessaire de télécharger 0 o/27,2 ko d'archives. Après dépaquetage, 106 ko seront utilisés.
Sélection du paquet dfu-programmer précédemment désélectionné.
(Lecture de la base de données... 115267 fichiers et répertoires déjà installés.)
Dépaquetage de dfu-programmer (à partir de .../dfu-programmer_0.5.4-1_amd64.deb)
...
Traitement des actions différées ("triggers") pour " man-db "...
Paramétrage de dfu-programmer (0.5.4-1) ...
```

Ensuite ces ligne de commande, effectuée avec le compte administrateur, ont permis la configuration grâce à un script python, où j'ai modifié le port d'utilisation puis exécuté le script.

```
root@iris-035:/home/iris/Documents# ./usb2ax run bootload.py
```

La première écrase le chip existant. Ensuite la seconde flash l'USB2AX. Puis la troisième commande démarre le nouveau chip.

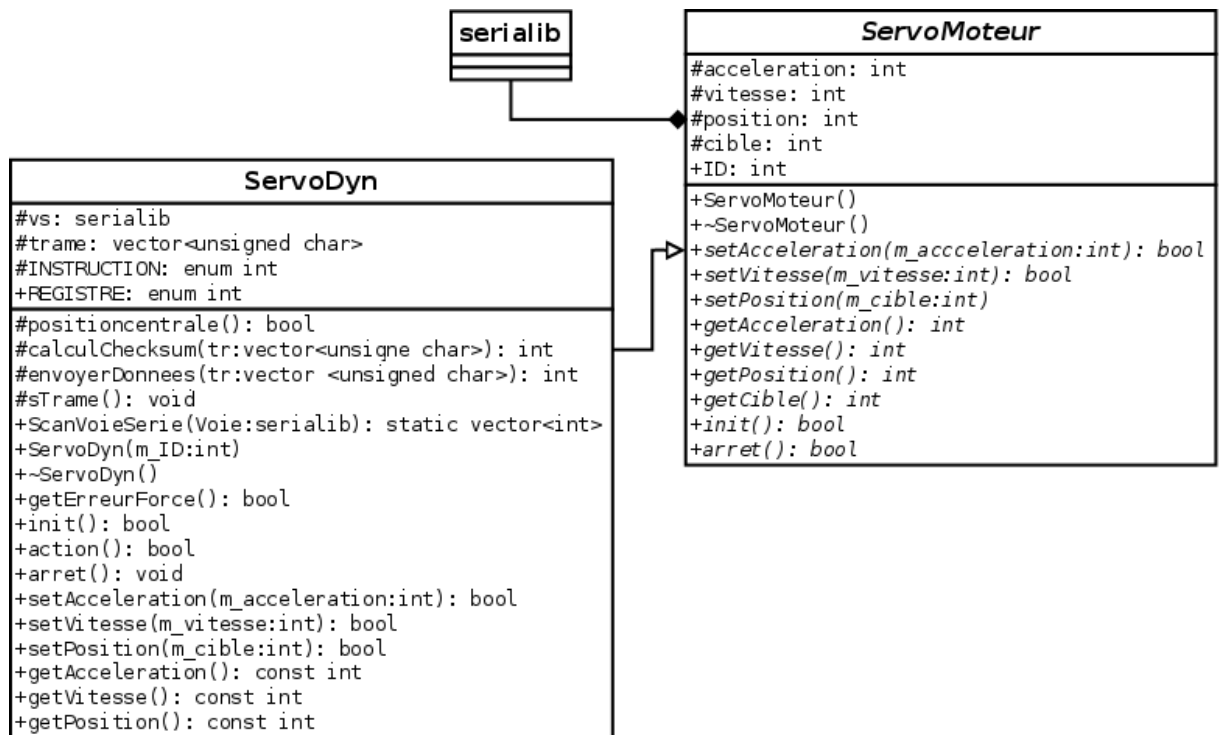
```
root@iris-035:/home/iris/Documents# dfu-programmer atmega32u2 erase
```

```
root@iris-035:/home/iris/Documents# dfu-programmer atmega32u2 flash USB2AX.hex
```

```
root@iris-035:/home/iris/Documents# dfu-programmer atmega32u2 start
```

4. Conception des classes ServoDyn et Neck

a. Classes ServoDyn et classe mère ServoMoteur



La classe **ServoMoteur** est une classe virtuelle, utilisée également pour les servomoteurs RC. Elle permet de définir les fonctions communes aux deux types de servomoteurs, RC et Dynamixel, qui sont toutes des fonctions virtuelles pures : elles ne sont pas implémentées dans la classe **ServoMoteur** mais dans ses dérivées.

« **serialib** » est la classe, libre de droit, utilisée pour la voie série. Toutes les informations sur cette classe sont sur le site dédié : <http://serialib.free.fr/html/index.html>

Elle a été choisie car elle permet grâce à des méthodes simples d'envoyer ou de recevoir des données sur la voie série.

La classe **ServoDyn**, dérivée de la classe **ServoMoteur**, instancie un objet **ServoDyn** correspondant au servomoteur auquel on s'adresse. Ensuite plusieurs opérations peuvent être effectuées sur le servomoteur tel que définir la vitesse de rotation avec « **setVitesse** » ou encore l'obtenir avec « **getVitesse** ».

La méthode « **positioncentrale** » permet de positionner le servomoteur au milieu de sa course, soit 180°.

La méthode « **sTrame** » crée le début de trame qui est identique à chaque fois. Ensuite la trame est créée dans les méthodes puis envoyée grâce à la méthode « **envoyerDonnees** ».

Toutes les méthodes envoyant des trames de positionnement sont dans le servomoteur concerné. Celui-ci se met en mouvement uniquement après l'appel de la fonction « **action** », qui permet de déplacer plusieurs servomoteurs en même temps.

Chaque trame se termine par le checksum, pour vérifier que la trame n'ait pas d'erreur de données ou de transmission. Pour cela on fait l'addition des octets, de celui du numéro de servomoteur jusqu'au dernier octet de données. Ensuite on effectue un ou exclusif avec le résultat de l'addition. Puis on fait un masque pour trouver l'octet de checksum.

Voici un algorithme expliquant le fonctionnement de la méthode « calculChecksum » utilisée dans la classe ServoDyn.

Algorithme calculChecksum

Variables :

nb, i : entier

tr : vecteur de caractère non signées

Début :

Pour (i allant de 2 à la taille de tr, i++)

Ajouter valeur de tr(i) à nb

Fin pour

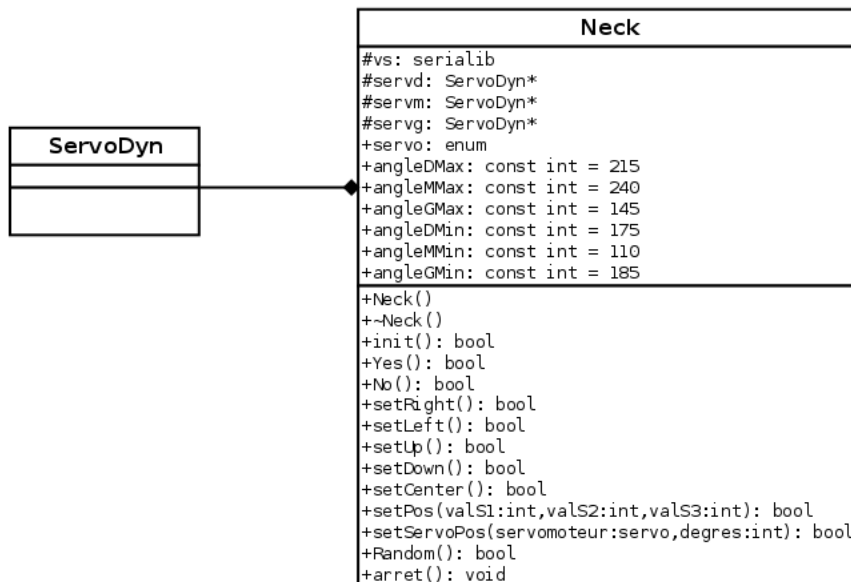
Affecter à nb : nb ou exclusif 0xFF

Affecter à nb : nb et 0xFF

Retourner nb

Fin

b. Classe Neck



La classe Neck est la classe utilisée par la classe principale du projet : Brain. Cette classe Neck permet d'effectuer des mouvements de la tête allant du plus simple à quelques-uns plus complexes. Elle est composée de la classe ServoDyn.

La fonction « init » de la classe permet d'initialiser les servomoteurs créés dans le constructeur. Elles leurs fixent une accélération et une vitesse rendant les mouvements plus fluides. Ensuite la tête se positionne de façon à regarder en face, sa position centrale.

Parmi les méthodes, la méthode « Yes » permet de faire le mouvement « oui » qui est un enchainement de mouvement de haut en bas. Ou encore le mouvement « non » avec la méthode « No ».

Les fonctions « setPos » et « setServoPos » sont des méthodes permettant de placer les servomoteurs dans une position entrée en paramètre.

« setPos » utilise les valeurs décimales comme paramètre.

« setServoPos » utilise des angles en degrés. La position centrale de la tête est considérée comme la position 0°. Le premier paramètre est un énumérateur permettant de choisir la direction : droite, gauche, haut ou bas. Le deuxième paramètre est l'angle pour tourner dans cette direction.

La méthode « Random » effectue un mouvement aléatoire de la tête à son appel. Elle est utilisée quand la tête n'est pas sollicitée pour montrer qu'elle est active et la rendre plus humaine.

Chaque méthodes, excepté la méthode « arrêt », renvoie un booléen pour vérifier la réussite ou non de la méthode.

c. Conclusion

Les classes effectuées sont fonctionnelles. Une fois intégrées, elles permettront d'utiliser la tête de manière autonome, tel que voulu. Une adaptation est envisageable pour faire fonctionner le suivi visuel d'une personne mais cette partie n'est pas encore intégrée.

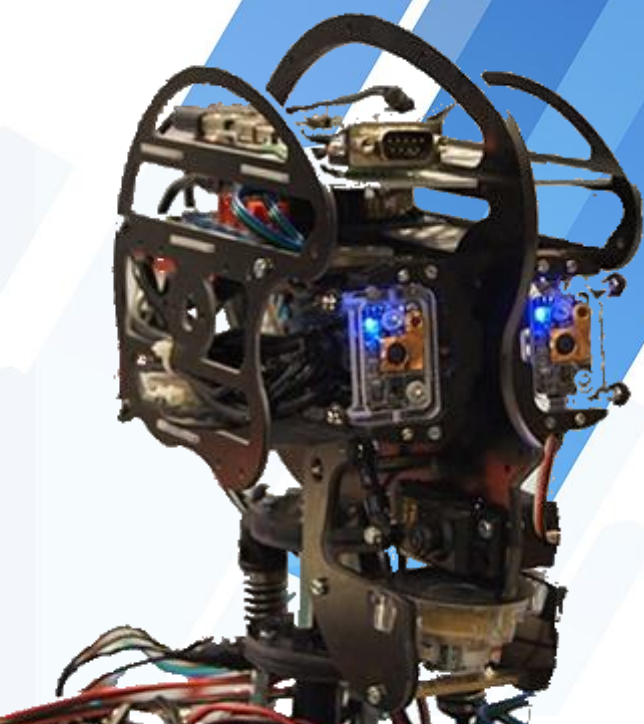
Webographie

-  Qu'est-ce que l'USB2AX ?
<http://www.xevel.fr/blog/index.php?post/2011/08/31/USB2AX>
-  Présentation de la MicroMaestro de Pololu :
<http://wiki.cybedroid.com/index.php?title=MicroMaestro>
-  Trame de contrôle de la MicroMaestro :
http://wiki.cybedroid.com/index.php?title=Protocole_maestro_Pololu
-  Présentation de l'adaptateur USB2Dynamixel :
http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm
-  Trame de contrôle des servomoteurs Dynamixel :
http://wiki.cybedroid.com/index.php?title=Protocole_dynamixel_Robotis
-  Présentation de CMUSphinx :
<http://cmusphinx.sourceforge.net/>
-  Présentation d'espeak et lien de téléchargement de celui-ci :
<http://espeak.sourceforge.net/>
-  Présentation de Mbrola et zone de téléchargement de différentes voix :
<http://tcts.fpms.ac.be/synthesis/mbrola.html>
-  Documentation d'OpenCV :
<http://opencv.org/>
-  Téléchargement de QT :
<http://qt-project.org/>
-  Documentation de Serialib et lien de téléchargement :
<http://serialib.free.fr/html/index.html>
-  Utilisation d'eSpeak et Mbrola en parallèle :
<https://csquad.org/2009/08/27/text-to-speech-avec-espeak-mbrola-et-speech-dispatcher/>
-  Redirection de la carte son sur la BeagleBone :
<http://situsavaislinux.blogspot.fr/2011/12/tutoriel-configurer-la-carte-audio-par.html>
-  Installation de dfu-programmer
<https://dfu-programmer.github.io/>

Annexes

Humanoïde « Aria »

**BTS IRIS
Lycée Turgot
Limoges**



2014 - 2015



